Managing Data Flow in Interactive MR Environments

Patrick Dähne Helmut Seibert
Computer Graphics Center (ZGDV)
Fraunhoferstraße 5
D-64283 Darmstadt, Germany
{patrick.daehne,helmut.seibert}@zgdv.de

ABSTRACT

In this paper the concept and design of a software framework which provides a transparent data flow for interactive Mixed Reality (MR) applications is discussed. The design was affected by our demands on platform independency, simplicity, network transparency, maximum performance and availability of runtime debugging facilities. Our software framework tries to simplify the development of MR applications by using the concept of a data flow graph. The developer builds such a graph from a library of small software components that communicate via the edges of the graph.

Keywords

Mixed Reality, Augmented Reality, Virtual Reality, Device Management, Tracking, Interaction.

1. INTRODUCTION

In our daily work on various Mixed Reality applications, we made the experience that we usually spend most of our time on implementing code that handles communication with hardware devices and software components provided by our project partners, instead of concentrating on the application itself. This paper describes the result of our efforts to minimize the work we have to do when implementing the "infrastructure" of our application.

The base of our MR applications is our self-developed rendering framework "Avalon" [3] that utilizes VRML/X3D. An interesting feature of VRML is that it does not only allow to describe graphical objects and animations, but also to specify the interactions that are possible with these objects. In fact, VRML allows to write huge parts of applications by using standard VRML nodes, JavaScript or Java, instead of doing tedious C++ compiler sessions.

Unfortunately, VRML has quite limited capabilities to integrate hardware devices and to communicate with other software components. In this paper we describe how we extended our rendering framework by an sophisticated device management system. The fundamental idea is to create a library of small software modules that handle specific tasks, for example operating devices or transforming data values provided by devices, and to assemble these modules into a data flow graph.

WSCG SHORT papers, ISBN 80-903100-9-5
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

2. RELATED WORK

The main research focus in the area of Mixed Reality has been the determination of the user's position and orientation (tracking), appropriate interaction methods for MR applications, graphical problems like the correct illumination of virtual objects and the occlusion of virtual objects by real objects, and the development of powerful and yet lightweight wearable computers. But recently, the development of MR software frameworks gained more and more attention.

An early example of a software framework is COTERIE [6]. COTERIE is a set of packages written in Modula-3 that allow to create "Distributed shared objects", i.e. data objects that are transparently shared by processes on different machines in the network.

Another well-known software-framework is Studierstube [9]. Studierstube is based on the concept of a distributed scene graph (Distributed Open Inventor), and a

data flow framework called OpenTracker [8] that handles different kinds of trackers.
DWARF [2] is using a concept similar to our own concept described in this paper. AR applications are build by using services that form a distributed data flow graph. CORBA is used for all network communication.
Tinmith [7] is using an object store based on Unix file system semantics. Applications can register callbacks on these objects to get notifications when an object changes. Objects are automatically distributed between different processes on the network.
Other software frameworks that focus on the development of VR applications are VR Juggler [4] and DIVERSE [1]. VRPN [11] and IDEAL [5] are device management systems that concentrate on the flexible and network-transparent access to VR devices.

3. SYSTEM DESIGN

Our concept of an MR framework is influenced by the design of VRML. In VRML, each node of the scene graph is a small status machine that receives events by inslots, changes its status according to the event, and sends events to outslots. Outslots and inslots are connected by ROUTEs. This architecture in fact establishes a second graph besides the scene graph, a data flow graph that contains large parts of the application logic.
The advantage of VRML over applications written in C++ or any other compiled language is that it is completely system independent. The same VRML world can be viewed on different operating systems, on low-end desktop PC's and high-end graphic workstations that operate immersive stereoscopic projection systems.
For our MR framework, we chose an approach that is strongly influenced by the VRML data flow graph and therefore has similar attractive properties. The main task of our MR framework is to handle input and output devices efficiently, because this is an area that is not handled by VRML. But besides simply allowing access to devices, our MR framework should also handle all preprocessing of the data provided by the devices. For example, it should not only allow the application to grab video frames from a web cam, but also to determine the position and orientation of the user by using these video frames (i.e. video based tracking).
Our intention is to create a library of small software modules that are specialized on simple tasks. Some software modules act as device drivers, i.e. they produce or consume data streams. Others act as filters, i.e. they transform incoming data streams. The modules are nodes of a data flow graph, and they receive data from and send data to other modules via the edges of the graph. The application developer should be able to build as much as possible of his application by simply assembling these prefabricated software modules into a data flow graph, allowing him to concentrate on the application logic.

3.1. Nodes

Nodes are the core component of the data flow graph. There are four types of nodes:

1. Nodes that produce data. These are usually device drivers of input devices, nodes that replay data streams stored on a hard disk, timers etc.

2. Nodes that transform data. Examples are nodes that transform coordinate systems, or video tracking systems that take video frames and transform them to position and orientation values.

3. Nodes that consume data. These are usually device drivers of output devices, nodes that store data streams on a hard disk, etc.

4. Nodes that do not deal with data in any way. This sounds strange at a first glance, but our system currently already has three nodes of this kind: The "Network" node that makes the system network transparent, the "Web" node that provides a user interface to the framework, and the "Inline" node that allows to integrate subgraphs stored in configuration files.

To create a new node, the developer has to derive a new class from the abstract node base class. Nodes can be linked into the application executable (mostly used for the basic, application-independent set of standard nodes provided by the system), or they can be loaded as plugins during runtime (application-specific nodes).
An important design decision that has to be made by the developer of a new node is whether the node uses its own thread or not. Device driver nodes for input devices usually always need to have an own thread, because they have to listen to the port the device is connected to. All other types of nodes do not necessarily need to have their own thread. For example, filter nodes that transform position values into another coordinate system usually directly calculate the new values when they are notified that new data is available. This means that they are in fact driven by the threads of the device driver nodes that provided the incoming data values. On the other hand, filter nodes that have to perform complex calculations, like video trackers, usually get their own threads.

3.2. Outslots and Inslots

Outslots and inslots are the means used by nodes to exchange data values. Outslots are used to send data to other nodes, and inslots are used to receive data from other nodes. Both outslots and inslots are typed, i.e. you have to specify what kind of data can be send to or received from the slot when you create it. When a

node writes data values into an outslot, the outslot automatically transfers copies of these data values to all inslots it is currently connected to. For efficiency reasons, smart pointers are used to transfer large amounts of data, i.e. copying data values into inslots usually just requires to copy a pointer.

3.3. Routes

When creating nodes of the data flow graph, the application developer has to specify a unique name for each of these nodes, e.g. "Joystick 1" and "Joystick 2" for two nodes that operate joysticks attached to the system. Each outslot and inslot of a node has a unique label as well, e.g. "Button #1" for the first button of a joystick or "X-Axis" for the x-axis. As a result, each slot in the system can be identified by a unique label consisting of the node name and the slot name, e.g. "Joystick 1/Button #1" for the first button of the first joystick or "Joystick 2/X-Axis" for the x-axis of the second joystick. To connect an outslot to an inslot, the application developer simply has to create a so-called "Route" that maps the label of an outslot to the label of an inslot. Of course, only slots sending and receiving the same data type can be connected by routes.

3.4. Configuration File

The configuration file allows to store the whole data flow graph on hard disk and to restore the graph when starting the application again. The format of the configuration file is XML due to the fact that this is a well-known, established standard for storing information.
Even though it is possible to create these XML files by hand, this approach is not recommended. Instead, the application developer uses the integrated user interface of the device management system to create a data flow graph that fulfills his needs. This user interface also allows to save the current status into a configuration file.
There are two ways to restore a saved graph from a configuration file:

1. The user simply loads the configuration by using the integrated user interface.

2. There is a special "Inline" node that allows to integrate configuration files into other configuration files, exactly the same way as the VRML "Inline" node allows to integrate several VRML subgraphs into one scene graph, or the C "include" preprocessor statement allows to integrate several pieces of source code into one file.

3.5. Network Transparency

The concept of a data flow graph makes it simple to introduce network transparency into the device management system. Nodes of the graph communicate via edges, so the most elegant solution is to allow edges to be created between nodes on different machines.
Our system does not support network transparency directly. Instead, it is made available by a special node called "Network". We did this to achieve a clear separation between the device management and the network code.
When the Network node starts operation, it automatically connects to all other Network nodes on the network. We are using a technique called "Multicast DNS" [12] to automatically discover all Network nodes available on the network without the need of any configuration or central servers. After connecting, the Network nodes create local proxy nodes for all nodes available on remote machines. These proxy nodes can be used to create edges between nodes on different machines.
Each data value that gets transferred via a network connection needs to be transformed into a system-independent byte stream (serialization). For each data type, a corresponding codec has to be implemented that performs this serialization. Such codecs already exist for the default data types provided by the system, but the application programmer has to implement and register codecs for all application-specific data types. Most codecs simply write the data values into the byte stream, but more sophisticated codecs are possible that compress the transferred data, e.g. when transfering video frames.

3.6. User Interface

Our device management system has an integrated Graphical User Interface (GUI) that allows to modify the data flow graph during runtime. During our work on several MR projects it soon became obvious that it is quite often impossible to do development and debugging directly on the machine the application is running on. Instead, we need means to control our application from other machines in the network.
Our solution is a small web server that provides a user interface build from HTML pages. This web server is not part of the device management system, instead it is implemented as a node that can be added to the data flow graph. The web server solution allows us to control the device management system from any device that has a simple web browser installed and that is connected via network to the machine the application is running on. The interface allows to inspect the current state of the system, to add and to remove nodes and routes from the graph, to change parameters, and to save the current state to or to restore it from a configuration file.

4. APPLICATION EXAMPLE

In this section we will discuss the solution for the task of combining tracking systems which is necessary in many Mixed Reality applications. The MEDARPA Project focused on the development of a flexible medical augmented reality system supporting minimal in-

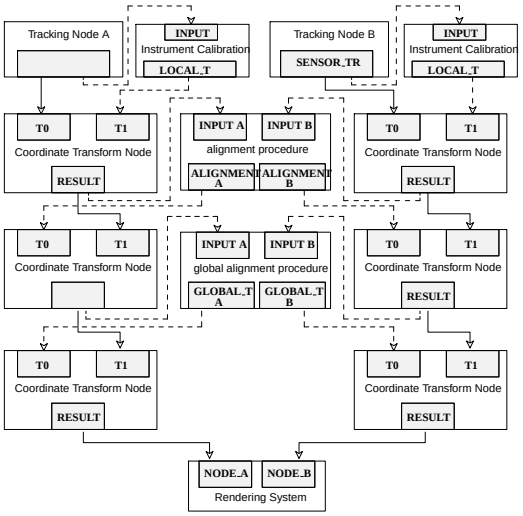

Figure 1. Graph for a composed tracking system. Dashed arrows depict event based data flow, solid arrows continuous data flow.

vasive interventions. Medical Image data and navigation support as augmentation are given on a transparent display which is mounted on a swivel arm. 6DOF tracking of the physicians head, the transparent display, patient and the instrument for the intervention is needed in order to provide the augmentations. An optical tracking system is used for the tracking of the physicians head as well as for the display position and orientation. For the tracking of the instrument an electromagnetic tracking system was chosen. These two tracking systems were combined as discussed in [10] by applying several consecutive transformations to all measurements to get the output of all tracking systems to a common coordinate system. A 6DOF transform consists of a rotation component and a translation component which are represented in a 4×4 transformation matrix and referred as T in the following. Usually the sensor of a tracking system cannot be placed exactly at the location which is to be tracked e.g. the tip of an instrument, the local transform T_l^{sens} allows to specify the location of the instruments tip in the sensor coordinate system. The alignment transform T_{tr}^{al} is needed to get the frame of reference defined by each tracking system aligned to a common frame of reference by mapping the output of one system to the frame of reference of the other system. A final transformation T_{al}^{w} allows to change the frame of reference if this is required by a special application.

For each measurement T_{sens}^{tr} reported by the tracking system, the transformation

$T = T_l^w = T_{al}^w \cdot (T_{tr}^{al} \cdot (T_{sens}^{tr} \cdot T_l^{sens}))$

needs to be calculated. The resulting transformation T can then directly be routed into the corresponding VRML Transform nodes of the scenegraph which is loaded on the Avalon rendering system. The resulting application dataflow is shown in figure 1.

5. CONCLUSION

In this paper we presented the framework we use for MR applications. It is currently implemented as a C++ library on several operating systems (Windows, MacOS X, Linux, IRIX and SunOS) and as a Java package (written in pure Java, therefore running on all systems where a Java virtual machine is available). We are currently developing MR applications based on the framework, and the experiences we gain from these practical trials propel the further advancement of the system. Our current research focus is the identification, design and implementation of new nodes that further ease the development of MR applications.

References

[1] L. E. Arsenault and J. Kelso. The DIVERSE Toolkit: A Toolkit for Distributed Simulations and Peripheral Device Services. In *VR 2002*, 2002.

[2] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riss, C. Sandor, and M. Wagner. Design of a component-based augmented reality framework. In *Proceedings of ISAR 2001*, 2001.

[3] J. Behr, P. Dähne, and M. Roth. Utilizing X3D for Immersive Environments. In *Proceedings of Web3D 2004*, pages 71–78, 2004.

[4] C. Cruz-Neira, A. Bierbaum, P. Hartling, C. Just, and K. Meinert. VR Juggler - An Open Source Platform for Virtual Reality Applications. In *Procs of 40th AIAA Aerospace Sciences Meeting and Exhibit '02*, 2002.

[5] T. Fröhlich and M. Roth. Integration of Multidimensional Interaction Devices in Real-Time Computer Graphics Applications. In *Computer Graphics Forum 19*, pages C–313 – C–319, 2000.

[6] B. MacIntyre and S. Feiner. Language-level support for exploratory programming of distributed virtual environments. In *Proceedings of UIST '96*, pages 83 – 95, 1996.

[7] W. Piekarski and B. H. Thomas. An object-oriented software architecture for 3D mixed reality applications. In *Proceedings of ISMAR 2003*, 2003.

[8] G. Reitmayr and D. Schmalstieg. An Open Software Architecture for Virtual Reality Interaction. In *Proceedings of VRST 2001*, 2001.

[9] D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Szalavari, L. M. Encarnação, M. Gervautz, and W. Purgathofer. The Studierstube Augmented Reality Project. *Presence*, 11, 2002.

[10] B. Schwald, H. Seibert, and M. Schnaider. Composing 6 DOF Tracking Systems for VR/AR. In *Proceedings of Computer Graphics International 2004*, pages 411–418, 2004.

[11] R. M. Taylor, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In *Proceedings of VRST 2001*, 2001.

[12] Multicast DNS, IETF draft. http://www.multicastdns.org/.