DECLARATION OF SCOTT DELMAN

I, Scott Delman, am over twenty-one (21) years of age. I have never been convicted of a felony, and I am fully competent to make this Declaration. I declare the following to be true to the best of my knowledge, information, and belief:

1. I am Director of Publications for the Association for Computing Machinery ("ACM").

2. Neither I nor ACM itself is being compensated for this Declaration.

3. Among my responsibilities as Director of Publications, I act as a custodian of records forACM's publications.

4. I make this Declaration based on my personal knowledge, information contained in the business records of ACM, or confirmation with other responsible ACM personnel with suchknowledge.

5. As part of its ordinary course of business, ACM publishes technical papers, including ACM'sconference proceedings, journals, and other full-text publications. ACM publications are distributed to ACM's individual and institutional subscribers, ACM members, and/or ACM conference attendees. Additionally, ACM publications are available for public download through ACM's Digital Library (as described further herein), which has existed since July 1997. Prior to the creation of the Digital Library in July 1997, members of the public could have obtained access to ACM publications by contacting ACM directly.

6. As part of its ordinary course of business, since July 1997, ACM has maintained a Digital Library where ACM publications are available to the public, either for free or purchase, depending on the article. The Digital Library includes keyword search functionality permitting users to search for articles using keyword search terms and to filter search results by whether the term appears in the title, full-text, index terms, and/or abstract of the article. In addition, users may search by author name, and may filter search results by the particular typeof journal, type of proceeding, and/or the date of publication.

7. In addition to providing access to ACM publications, the Digital Library also indexes papersby other publishers. When a non-ACM publication is indexed in the Digital Library, the public may access "metadata" concerning the publication through the Digital Library but cannot access the full text of the paper; instead, a member of the public who wishes to obtainsuch a paper would have to contact the publisher of that paper directly.

8. *"An Open Software Architecture for Virtual Reality Interaction"* by Gerhard Reitmayr and Dieter Schmalstieg on March 25, 2002, in Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '01). Association for Computing Machinery, New York, NY, USA, 47-54. DOI:https://doi.org/10.1145/505008.505018. Exhibit 1 is a true and correct copy of this paper.

9. The conference started on November 15, 2001, and the paper would have been available to the conference attendees on this date.

10. I hereby declare that all statements made herein of my own knowledge are true and that allstatements made on information and belief, are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001.

I declare under penalty of perjury that the foregoing statements are true and correct.

Executed on: January 28, 2022

Scott Delman, Director of Publications

Exhibit 1

An Open Software Architecture for Virtual Reality Interaction

Gerhard Reitmayr
Vienna University of Technology
Favoritenstraße 9-11/188/2
A1040 Vienna, Austria
reitmayr@ims.tuwien.ac.at

Dieter Schmalstieg
Vienna University of Technology
Favoritenstraße 9-11/188/2
A1040 Vienna, Austria
schmalstieg@ims.tuwien.ac.at

ABSTRACT

This article describes OpenTracker, an open software architecture that provides a framework for the different tasks involved in tracking input devices and processing multi-modal input data in virtual environments and augmented reality application. The OpenTracker framework eases the development and maintenance of hardware setups in a more flexible manner than what is typically offered by virtual reality development packages. This goal is achieved by using an object-oriented design based on XML, taking full advantage of this new technology by allowing to use standard XML tools for development, configuration and documentation. The OpenTracker engine is based on a data flow concept for multi-modal events. A multi-threaded execution model takes care of tunable performance. Transparent network access allows easy development of decoupled simulation models. Finally, the application developer's interface features both a time-based and an event based model, that can be used simultaneously, to serve a large range of applications. OpenTracker is a first attempt towards a "write once, input anywhere" approach to virtual reality application development. To support these claims, integration into an existing augmented reality system is demonstrated. We also show how a prototype tracking equipment for mobile augmented reality can be assembled from consumer input devices with the aid of OpenTracker. Once development is sufficiently mature, it is planned to make Open-Tracker available to the public under an open source software license.

Keywords

Tracking, Mobile Augmented Reality, Virtual Reality, XML

1. INTRODUCTION

Tracking is an indispensable part of any Virtual Reality (VR) and Augmented Reality (AR) application. While the need for quality of tracking, in particular for high performance and fidelity, have led to a large body of past and current research, little attention is typically paid to software engineering aspects of tracking software. Some current systems have a modular approach that allows to substitute one type of tracking device for another. Typically, this is the approach taken by commercial VR products that offer turn-key support for many popular tracking and input devices, but at the cost of a limited amount of extensibility and configuration options. In particular, they make it hard to combine existing features in novel ways.

In contrast, research systems may offer features not found in commercial systems, such as prediction or sensor fusion, but are usually limited to their particular research domain and not intended for the end user. In such systems, replacing a piece of hardware or changing its configuration usually leads to rewriting a significant portion of the tracker software.

In the middle(-ware), there is a lack of tools that allow for a high degree of customization, yet are easy to use and to extend. One notable exception is the MR toolkit [21] of the University of Alberta, which still serves as a starting point for many VR research projects despite its aged architecture and lack of active development. What is needed is a system that allows mixing and matching of different features, as well as simple creation and maintainance of possibly complex tracker configurations.

In this article, we describe a tracking software system called *OpenTracker* with the following characteristics:

- An object-oriented approach to an extensive set of sensor access, filtering, fusion, and state transformation operations
- Behavior specification by constructing graphs of tracking objects (similar in spirit to scene graphs or event cascades) from user defined tracker configuration files
- Distributed simulation by network transfer of events at any point in the graph structure
- Decoupled simulation by transparent multi-threading and networking
- A software engineering approach based on XML [4], which allows to use many generic tools such as [2, 11, 10] for development, documentation, integration and configuration
- An application independent library to be integrated into software projects

VRST'01, November 15-17, 2001, Banff, Alberta, Canada.

Through its scripting capability (tracker configuration files) as well as easy integration of new tracking features, *OpenTracker* encourages exploratory construction of complex tracking setups. It is equally useful for end users, which can fully exploit their hardware without any custom programming, as well as developers, who can easily build test environments. The modular approach gives instant access to wide range of tracking related functionality for any application. Through the release under the LGPL Open Source license [7], *OpenTracker* is available to a larger audience.

2. RELATED WORK

Ideas implemented in *OpenTracker* were drawn from several areas:

Device abstraction is a standard requirement for 2D graphical user interfaces, (e. g. GKS [12]), and sometimes incorporated into 3D applications [9]. There is a number of libraries such as VRPN [15], MRToolkit [21] implementing device abstraction for input devices typically found in VR and AR systems. Their main goal is to provide a fixed interface to the application for different devices and provide simple services for relaying the data over the network between several hosts. However, these libraries mostly lack any further means to process the data. Device abstraction is also an important goal of *OpenTracker*. However, it goes beyond pure abstraction using a static interface in that the data can be re-combined in novel ways.

Many interactive systems employ sophisticated event handling schemes. State changes to attributes of scene objects are either propagated through functional dependencies (e. g. routes in VRML [5], engines in Open Inventor [22]), or may be handled by user supplied callback functions (e. g. script nodes in VRML [5]). These approaches inspire the architecture of *OpenTracker*, although none of them deals specifically with tracker configurations.

Finally, an important requirement for virtual environments is support for distributed simulation, partly to support simultaneous users, partly to better exploit available hardware. Decoupled simulation was first introduced in MR [21], and later used in almost any major VR software system. Decoupled simulation can either be implemented by multithreading and/or symmetric multiprocessing on one host, or by configuring a small set of hosts to work as an ensemble. The latter approach may be inferior performance-wise because of network lag, but it is inexpensive and flexible, and thus favored by many researchers - for example, Rekimoto's "hyperdragging" system [19] uses a distributed architecture very much like our own.

3. DATA FLOW OF TRACKING DATA

In a typical VR or AR application tracking data passes through a series of steps. It is generated by tracking hardware, read by device drivers, transformed to fit the requirements of the application and send over network connections to other hosts. Different setups and applications may require different subsets and combinations of the steps described but the individual steps are common among a wide range of applications. Examples of such invariant steps are geometric transformations, Kalman filters and data fusion of two data sources.

The main concept behind *OpenTracker* is to break up the whole data manipulation into these individual steps and build a data flow network of the transformations. To describe the details of this concept, we will need some theoretical definitions which are discussed in section 3.1. Details of an actual implementation are described in section 3.2.

3.1 Data Flow Concept

Each transformation is represented by a node in a data flow graph. Nodes are connected by directed edges to describe the direction of flow. The originating node of a directed edge is called the child whereas the receiving node is called the parent. To allow more than simple linear graphs, we introduce the following concepts.

Multiple Input Ports and References

Each node has one or more input ports and a single output port. A port is a distinguished connection point for an edge, i.e. the node can distinguish between events passing through different node ports. The output port of one node is connected to any of the input ports of another node. This establishes the flow by defining directed edges in the graph. A node receiving a new data event via one of it's inputs computes a new update for itself and sends the new data event out via its output port.

Multiple input ports are desirable because computations typically have more than one parameter. Dynamic transformations, for example, are parameterized by the value of another node and thus use the data value received by a child to be transformed differently from the data of the parameterizing child. Merge nodes may select part of the data of an event based on the input port the event used. This allows more complex computational structures.

Additionally, an input port can be connected to several output ports. This enables several children nodes connected to the same input port of a node. Upon receiving an event, the parent node can only distinguish between the input ports, not between the actual children.

Conversely, an output port can also be connected to other nodes by using references within the graph. This establishes new edges between a nodes output port and other nodes input ports. However this is transparent to the child node. It cannot selectively send events to only one parent, but all events are distributed equally to all parents.

Edge types

The basic mechanism behind the data flow concept is event passing. Data events are passed from the children nodes upward to their parents. However, not all computations fit well into this model: Algorithms that operate on a vector of tracker measurements or that require or compute the tracker state at an arbitrary point in time require different types of input or output interfaces. Examples are smoothing algorithms that take a history of events into account, or prediction algorithms that compute an expected measurement for a given point in time.

Therefore, we also distinguish between different edge types. Edges are typed by typing the ports of the nodes they connect. We establish the rule that only two ports of the same type can be connected and this type is then the type of the edge. There are three edge types: *event*, which is implemented by event passing, *event queue* and *time dependent*. The latter two are implemented as interfaces that are polled by the parent node, because the data returned is parameterized. In the case of the *event queue* interface, it is possible to query the number of stored events and retrieve them by index. The *time dependent* interface can be queried by spe-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.