OpenTracker – An Open Software Architecture for Reconfigurable Tracking based on XML

Gerhard Reitmayr
Qualcomm
136 PUBLICATIONS **3,332** CITATIONS
SEE PROFILE

Dieter Schmalstieg
Graz University of Technology
490 PUBLICATIONS **9,769** CITATIONS
SEE PROFILE

OpenTracker - An Open Software Architecture for Reconfigurable Tracking based on XML

Gerhard Reitmayr and Dieter Schmalstieg
Vienna University of Technology
{reitmayr|dieter}@cg.tuwien.ac.at

Abstract

This paper describes OpenTracker, an open software architecture that provides a generic solution to the different tasks involved in tracking input devices and processing tracking data for virtual environments. It combines a highly modular design with a configuration syntax based on XML, thus taking full advantage of this new technology. OpenTracker is a first attempt towards a "write once, track anywhere" approach to virtual reality application development. To support these claims, integration into an existing augmented reality system is demonstrated. Once development is sufficiently mature, it is planned to make OpenTracker available to the public under an open source software license.

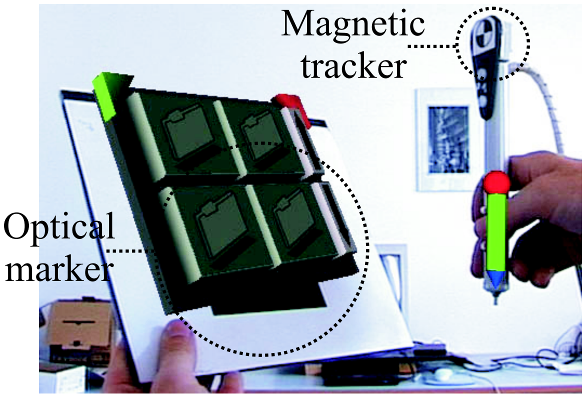

Figure 1. Different tracking technologies integrated transparently by OpenTracker

1. Introduction

Tracking is an indispensable requirement for all kinds of virtual reality (VR) and augmented reality (AR) systems. While the quality of tracking, in particular the need 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 solutions may 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 extendability 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 means 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 [14] 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 maintainence of possibly complex tracker configurations.

In this paper, we present 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 tracker state at any point in the graph structure

- Decoupled simulation by transparent multi-threading and networking
- A software engineering approach based on XML [6], which allows to use many generic tools such as [2, 9, 10] for development, documentation, and configuration.

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. In brief, *OpenTracker* attempts to provide a ”write once, track anywhere” paradigm similar to the spirit of the Java programming language. Through the release under a public domain license, we plan to make *OpenTracker* available to a larger audience in the near future.

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 [11]), and sometimes incorporated into 3D applications [8]. 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 [7], engines in Open Inventor [15]), or may be handled by user supplied callback functions (e. g. script nodes in VRML [7]). 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 [14], and later used in almost any major VR software system. Decoupled simulation can either be implemented by multi-threading and/or symmetric multiprocessing on one host, or by configuring a small set of hosts to work as an ensemble.

3. Scope of the software architecture

While some approaches to general event architectures exist (e. g. [5]), the current scope of OpenTracker are traditional VR applications. It thus deals primarily with position and orientation information (six degrees of freedom, 6DOF), although some other event types such as button events and 2D position information (such as from a desktop mouse) are supported. While it is straightforward to cover other data types (such as input from a pressure sensitive tablet), we choose to limit ourselves to the domain immediately useful for VR applications.

The software is designed as a class hierarchy of tracker objects, implemented in C++. Every tracker object defines an interface that can answer a query for the current position and orientation as well as the state of the associated buttons. At runtime, these tracker objects are assembled into a directed acyclic graph (DAG) - or frequently, a set of DAGs - according to the instructions in a user-supplied configuration file written in XML. We distinguish *source objects*, which are leaves in the graph and receive their data values from external sources, *filter objects*, which are intermediate nodes and modify the values received from their child nodes, and *sink objects*, which propagate their data values received from their child nodes to external outputs.

3.1. Source objects

Most source objects encapsulate a device driver that directly accessess a particular tracking device, such as a Polhemus or Ascension tracker connected to a serial interface. Other source objects form bridges to complex self-contained systems, such as the video tracking library from ARToolKit [12]. Yet other source objects emulate tracker via the keyboard or simply respond with constant values (useful for development and debugging) or access network data (see section 4).

Some source objects have a multi-threaded execution model to implement a decoupled simulation model [14] (e. g., when blocking I/O must be used).

3.2. Filter objects

Filter objects have one or more children. When queried, filter objects pass on the query to determine the state of their children, then compute their own state based on the returned data. A non-exhaustive list of filters includes:

- Transformation filters perform geometric transformations of their childrens values. These include pre- and post-transformations and may be static or depend on data values received from other children. The latter allows to modify the filtered state relative to another tracker state.
- Prediction filters allow to partially compensate for lag in the measuring and processing tracker data.
- Noise and smoothing filters are handy to deal with inherent inaccuracies of trackers.

- Undistortion filter are necessary e.g. to linearize distortions in the magnetic field of a magnetic tracking device.
- Permutation filters are necessary to match data representations from different hardware or software platforms, such as equivalent, but incompatible quaternion representations.
- Merge filters assemble new data values using different parts of the data values of several children. Sample uses include the combination of orientation from an inertial tracker with position information from an acoustic tracker, or adding a button device to a closed tracking solution such as Polhemus Ultratrak.
- Conversion filters are able to translate one data type into another. For example, 2D positions from a desktop pointing device can be translated into 3D positions by adding a constant third value.
- Clamp filter are special nonlinear transformation filters that cut off values at user-specified extrema, for example to deliberately limit interaction to a valid range.
- Store-and-forward filters are useful if transient loss of tracking can be expected, for example if occlusion occurs in optical tracking. The last measured value is simply repeated to provide at least a reasonable and valid state.
- Confidence filters select data values from different children based on some measure of confidence in the accuracy of the data.

3.3. Sink objects

Sink objects are similar to source objects but distribute data rather than receive it. They include output to network multicast groups, debugging output to a user interface or shared memory to integrate *OpenTracker* as a library into other applications.

The presence of sink object drives the evaluation model of *OpenTracker*. All sink objects in a tracker object graph are registered upon creation, and their respective state evaluation method is triggered periodically. We found this to be more effective than a pure client-driven lazy evaluation scheme, as it avoids potentially costly recomputation of intermediate values for every invocation.

4. Distributed tracking

There are several reasons why is is desirable to share tracker data over a network:

- Using the tracker data at multiple host computers for a distributed virtual environment (local or remote): Input in the form of tracker data becomes readily available through transparent network access via *OpenTracker*. The scene database still has be to kept consistent through a proprietary application protocol, but the task is much simplified.
- With the same approach, multi-processing based on inexpensive PCs becomes possible with little configuration effort. This is useful to achieve some degree of load balancing. In particular, computationally expensive functions such as filtering or undistortion can be assigned to either sender or receiver, depending on the computational budget.
- Network support makes it easy to span multiple operating systems, in particular if a specific tracking device or service is only available at one particular host (e.g., an SGI O2 has fast video hardware but a slow CPU, whereas for a PC the opposite may be true).

OpenTracker allows multiple senders and receivers of tracker data to communicate asynchronously through the use of IP multicasting (Figure 2). This approach effectively implements decoupled simulation in a larger scope, since each of the senders and receivers can operate independently. It is even possible for a single host to operate as a sender and receiver at the same time, by picking up data, then modifying it and re-sending it to the network on another network channel.

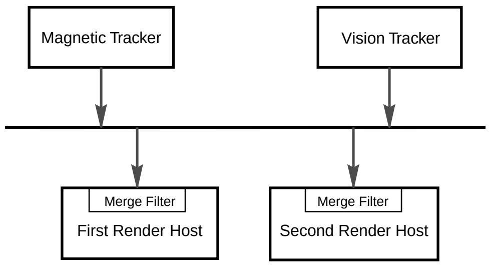

Figure 2. Distributing tracking data to different rendering hosts

While there is a prefered network protocol for *OpenTracker*, several formats are understood. In the following, we give some examples as to how a networked setup can be used:

- A tracker server (typically a cheap PC with lots of serial I/O boards running Linux) samples an Ascension Flock of Birds at highest rate and sends the resulting data stream via multicast to several clients using this data to animate a collaborative virtual environment.

- The Polhemus Ultratrak uses a proprietary network format and IP unicast packages. Unfortunately, its closed architecture does not support input devices with buttons such as a stylus or 3D-mouse. Therefore, we added a tracker object to the client that is able to decode the Ultratrak protocol. A button source reads button values from a standard parallel interface, and a merge filter combines these two sources to emulate a complete VR input device.

- A combination of vision tracking and magnetic tracking – see section 6 for details.

5. Software engineering with XML

XML, the extensible markup language, is the emerging standard for web-based applications and software systems [6]. XML is a markup definition language that allows to define hierarchical markup languages with so-called document type definitions (DTD). With the appropriate DTD, standard XML tools can be used to conveniently edit, type check, parse, and transform any XML file.

Thus, providing a simple DTD for describing hierarchies of tracker objects opens access to software libraries and tools that simplify several steps of the development cycle:

- A DTD editor can be used to design and maintain the DTD.

- An XML parser enforces content format on the tracker configuration file while building the corresponding structure in memory, thus automatically performing many of the consistency checks that have otherwise to be hand-coded.

- A convenient XML editor with a graphical user interface allows the end user to design the tracker configuration without having to master the syntax.

- Using the extendible style language (XSL) [1], automatic textual and even graphical documentation can be created from a tracker configuration file, for example by using the free graph drawing utility *dot* [4] (see Figure 3).

Markup languages are generally used to annotate textual documents with a structural information. Thus a general XML document consists of text grouped and structured with tags. Markup languages defined in XML consist of elements, essentially expressed as tags, and a structural model (the content model) of the possible ways these elements may be nested. Moreover, elements are annotated by name - value pairs called attributes.

OpenTracker maps elements to objects and attributes to members of these objects. We are not using any textual content but purely rely on the content model provided by the DTD. An open source XML parser [2] builds a tree of elements representing the given configuration file. *OpenTracker* walks the tree and creates a new object for each element based on the elements name. The string values of the attributes are parsed according to the objects class and the corresponding members are set. Attributes typically describe such data as the parameters of a transformation.

The DAG structure is created by using unique IDs on elements and referencing these IDs in placeholder elements. Again XML enforces the uniqueness of these IDs and the parser library simplifies the search for the referenced elements.

Restrictions on the number of children and the possible types are described in the DTD. *Source objects* typically do not have any children objects as they rely on data from external sources to compute their own data. A number of *filter objects* get the value of a single child object, transform it and pass it on. Confidence filters use any number of children to compute their data value. The data of the different children enters in the same way into the computation.

In another case different children objects influence the computation in different ways. Dynamical transformations, for example, are parameterized by the value of another object and thus use the data value of the object to be transformed differently from the data of the parameterizing object. This is handled by using wrapper elements. An object requiring marked children is mapped to an element that may only have certain marker elements as children. These marker elements in turn may have any other element as child again. The marker elements are mapped to marker objects, that perform no special function and return the value of their child object. They can be queried by the filter object to derive how to use this value.

The following XML code describes a server configuration for a single tracked pen using the ARToolKit video tracking library:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--simple test configuration for the
    ARToolKit tracker server system -->
<!DOCTYPE TrackerServer SYSTEM "opentracker.dtd">
<TrackerServer>
  <configuration>
    <Network name="ARToolKit"
             multicast-address="224.100.200.101"
             port="6666"
    />
    <Video camera-parameter="camera.dat"/>
    <ScreenSink headerline="ARToolKit Server"/>
  </configuration>
  <Station name="pen" number="1">
    <ScreenOutput comment="Pen station">
      <Transformation translation="0 0 -0.5"
                      scale="0.001 0.001 0.001">
        <Marker tag-file="pen.tag"
                vertex="-36 -36 36 -36 36 36 -36 36"
        />
      </Transformation>
    </ScreenOutput>
  </Station>
</TrackerServer>
```

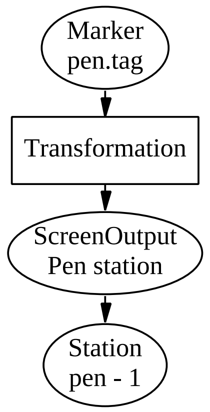

Figure 3. Graph visualization of the example configuration

6. Results and examples

We have successfully used *OpenTracker* in a number of experimental setups either using it as our sole source of tracking data or integrating it with an existing setup.

For example, in an experimental pen-and-pad interface, we combined a vision tracking approach (ARToolKit) for the pad with a magnetic tracker (Ascension Flock of Birds) for the pen. Two separate servers for video and magnetic tracking were sending their measurements over the network to a rendering host, where the combined data was picked up by an *OpenTracker* component (Figure 2). The tracking data from this source was transformed to register with the tracked objects and fed into *Studierstube*, an augmented reality environment recently described in [13]. The rendering was then overlayed onto the video input from the camera. Figure 1 shows the pip and a simple pen.

In this setup there are two tracked objects, a pen and pad tracked magnetically and by video, respectively. The data of both is automatically merged and transformed by *OpenTracker*. Then it is distributed on the network and simultaneously displayed on the screen for debugging purposes. This behaviour was solely defined by the configuration file.

7. Conclusions and future work

None of the important properties of *OpenTracker* – such as filtering, decoupled simulation, or configuration languages – are genuinely new. Yet we were suprised in being unable to find a publicly available solution truly suited for the needs of a virtual reality developer – a lack which led to the conception of *OpenTracker*. While to capabilities of *OpenTracker* are uttlerly unspectacular, we found them much needed.

Much remains to be done. The current version 0.1 is all but complete. We thus invite contributors all over the world to help improving this open source project.

For more information, check out the project home page: http://www.cg.tuwien.ac.at/research/vr/opentracker/

Acknowledgments

This project was sponsored by the Austrian Science Fund *FWF* under contract no. P-14470-INF. Special thanks to Anton Fuhrmann for his perpetual will to comment, to Roman Rath for his contributions to the implementation, and to Mark Billinghurst for ARToolKit.

References

[1] S. Adler et al. Extensible stylesheet language (XSL) 1.0. http://www.w3.org/TR/xsl/.
[2] Apache. Xerces XML parser. http://xml.apache.org/xerces-c/index.html.
[3] P. Appino, J. Lewis, L. Koved, D. Ling, D. Rabenhorst, and C. Codella. An architecture for virtual worlds. *Presence: Teleoperators and Virtual Environments*, 1(1):1–17, 1992.
[4] AT&T. Graphviz. http://www.research.att.com/sw/tools/graphviz/.
[5] F. Behmaram-Mosavat and L. M. Encarnao. A software framework for user-centered multi-modal interaction. *CG topics, INI-GraphicsNet, Darmstadt, Germany*, 12:5–6, 2000.
[6] T. Bray, J. Paoli, C. M. Sperberg-McQueen, et al. Extensible markup language (XML) 1.0. http://www.w3.org/TR/REC-xml/.
[7] R. Carey and G. Bell. *The Annotated VRML 2.0 Reference Manual*. Addison-Wesley, 1997.
[8] T. He and A. Kaufman. Virtual input devices for 3D systems. In *Proc. IEEE Visualization'93*, pages 142–148. IEEE, 1993.
[9] IBM. Xeena XML editor. http://www.alphaworks.ibm.com/tech/xeena.
[10] Icon Information Systems GmbH. XMLSpy. http://www.xmlspy.com.
[11] ISO. Graphical kernel system (GKS). IS 7942, 1985.
[12] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99), San Francisco*. IEEE, October 1999.
[13] D. Schmalstieg, A. Fuhrmann, and G. Hesina. Bridging multiple user interface dimensions with augmented reality. In *Proceedings of the 3rd International Symposium on Augmented Reality (ISAR 2000), Munich, Germany*. IEEE, October 2000.
[14] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled simulation in virtual reality with the mr toolkit. *ACM Transactions on Information Systems*, 11(3):287–317, July 1993.
[15] P. Strauss and R. Carey. An object oriented 3D graphics toolkit. In *Proceedings SIGGRAPH'92*. SIGGRAPH, 1992.