A Flexible Prototyping Tool for 3D Real-Time User-Interaction

Roland Blach, Jürgen Landauer, Angela Rösch, Andreas Simon

Competence-Centre Virtual Reality
Fraunhofer Institute for Industrial Engineering (IAO)
Nobelstr. 12, D-70569 Stuttgart, Germany
phone: +49-711-970-2153, fax: +49-711-970-2213
Roland.Blach@iao.fhg.de
http://vr.iao.fhg.de

Abstract

High interactivity in real-time environments requires new system design concepts for virtual reality environments. We will define our understanding of a modern system and describe the prototypical system *Lightning*, which addresses some of these new features.

Keywords: Virtual reality, Virtual environments, Virtual reality development system, Immersive environments, Interactive real-time systems

1.0 Introduction

Virtual Reality (VR) in our understanding, is multimodal interaction with dynamic and responsive computer generated or so-called virtual environments. The main focus is on interaction, which combines adequate presentation of the environment with its manipulation. Multimodality defines the more hardware oriented interface definition as the combination and cooperation of various input and output channels like speech, gesture, sound, position, video, etc. A highly interactive system has to deal with time in general and specifically with variable time frames and their synchronization. Responsive virtual environments should operate in real-time, that is, the response time and update rate of the system is high enough that it generates an experience of continuity. Continuity is a major precondition for the impression of an imperceptible boundary between user and virtual environment, the so-called interface. We consider this property to be one of the major differences between VR-systems and other 3d-systems such as CAD systems.

These techniques serve one main purpose: the enhancement of human computer interaction. Especially in problem domains of high complexity the use of immersive virtual environments promises a better insight. Immersion in our opinion is a product of the used techniques and surrounds the user with the computer generated world, which seems to be a different experience compared to classical interfaces. Obvious examples

are complex evaluation or planning tasks like architecture or design, medical training, fluid dynamics in engineering or assembly planning, etc..

2.0 Basic Considerations

The user interface of a virtual reality system (VR-system) can be considered as a closed loop system where the user is an integral part of the system (‘human in the loop’). The generalization of the user concept to other external systems including also more users seems obvious and useful.

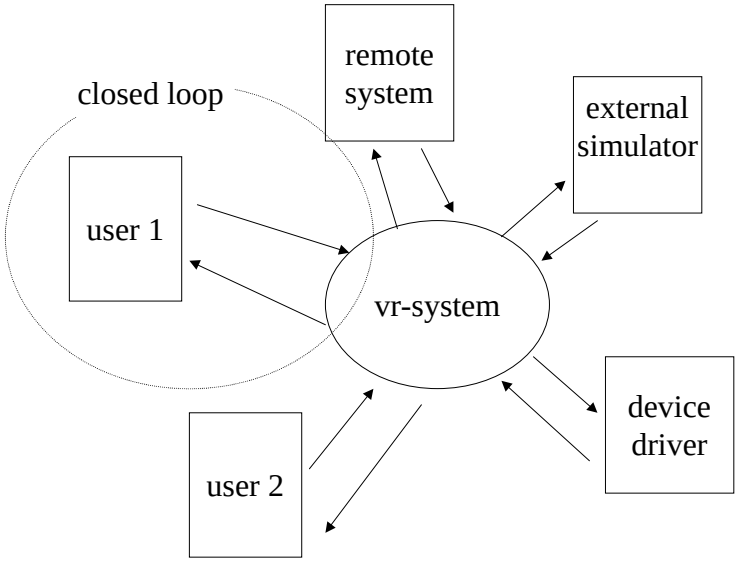

Fig 1. General VR-System Structure

A VR-system can be divided in internal and external subsystems. As external we consider all subsystems where only the interface is known but there is no complete control in the sense of parameterization or restructuring during run-time. Obvious examples are human users, external simulation programs, hardware devices, etc.

The internal simulation system contains adequate abstractions of entities and their communication system to define the application and input/output modules. Basically it samples the outside world via sensors or other communication mechanisms. The next time-step will be computed according the inbuilt dynamics, behavior or interaction rules. The display modules sample the internal state of the simulation and generate the adequate output.

We see some similarities between complex artificial machines/organisms like robots and VR-systems. A minor difference is actually the embedding factor of reality and virtuality. In classical virtual reality systems the user is surrounded by computer generated environments. The VR-system has to capture ‘only’ the user, but has to generate the impression of a complete and convincing environment. A robot has to capture its physical environment and to compute ‘only’ the adequate output for its effectors. A major difference is the existence of a goal of the system. A virtual reality system has no defined end state or goal, it is a kind of open system.

Some researcher have defined performance goals like frame rate and used control theory methodologies [13] to obtain these goals. Basic approaches are realized in visual rendering as for example frame-rate based level of detail (LOD) switching to achieve a constant frame rate, which is only a small part of the overall performance. Another approach are prediction schemes for motion tracking [14] where for example Kalman filtering was proposed.

This more general understanding of a virtual reality system leads us to the assumption that some of the problems might have been solved already in other research areas as for example in robotics, control and system theory or artificial life. Especially strategies to overcome the lack of information in case of sensors are needed. General synchronization schemes might be derived with these strategies.

One of the challenges for system design is a flexible and extensible methodology of achieving consistency between known and unknown input and output modules and the internal simulation. We have concentrated in our concepts basically on space and time consistency which leads to the problems of synchronization and exact overlay of physical and virtual space. General consistency should consider basically all state information, as for example colour space matching is an important issue for augmented reality.

A virtual reality system distributes output, for example visual rendering and input tasks for example collecting position tracking data to independent process to be able to control request times. The application consists often of an event propagation module and sometimes if higher computing load is expected, independent simulation processes, for example a dynamic simulation of moving objects. But even today many systems do have an inflexible structure concerning the distribution of the application load in regard to heavy interactive user involvement. Main focus was the real-time capability of the display output, or mostly the visual rendering, although some researchers are concentrating also on other features like 3D acoustics [1] or force-feedback output [2][7]. Input processes run usually as an external process, as fast as the hardware device is able to deliver the data. In our understanding it is not just necessary to collect input-data in real-time, it has to be evaluated in real-time. Output should sample the internal real-time simulation at a physiological suitable and necessary rate. That implies a shift to a real-time input loop compared to today's systems where the focus is more on real-time output. Two main points have to be addressed:

- Real-time capability of the system behavior, that is evaluation and event propagation of the input data completely independent from output.
- Internal storage of sampled input data to produce continuous output at a later time.

This real-time capability is particularly interesting in case of dynamic behavior of objects in combination with user interaction.

In the following we will describe how our prototype *Lightning* addresses these considerations.

3.0 *Lightning*

The *Lightning* virtual reality system was first presented in 1996 [6] as a rapid prototyping tool for VR applications, particularly in the architectural and presentation domain. Since then it has evolved to a mature tool for interactive engineering environments, where prototyping of 3D user interfaces becomes an important issue.

3.1 Software architecture overview

The design of the system is inspired by a common event propagating paradigm similar to for instance VRML 2.0 [11] or Open Inventor [12], as it can be seen in the following section. Fig. 2. gives an overview of its system architecture.

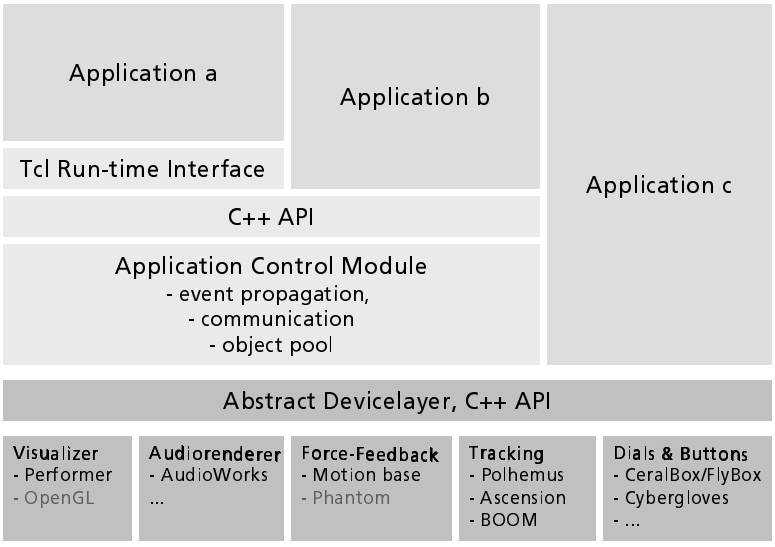

Fig 2. *Lightning* overview

Its core component is a database called object pool. Application developers define the environment merely by creating or deleting objects within this database. The interactivity and the behavior can be introduced with function objects and communication channels, so-called routes. These communication channels define application specific event propagation and therefore the interactivity.

A basic object type is provided with several properties:

- Unified Interface slots so-called fields provide a standardized interface for the event propagation module.
- A uniform update function, which changes the internal state according to the data at the input fields and the internal state of the object.
- Execution of this update function, only if the input data has changed or the object has been handed to a special administration module, which always calls the update function. This is a generalized implementation of the so called sensor type in VRML2.0 [11].

Objects have a certain internal linkage, which connects them to the *Abstract Device Layer*. System or application developers can extend the system simply by providing a new object with the described field interface.

Generic render devices exist for visual and audio renderer. The specific implementation will be filled in via inheritance, which in case of the visual renderer provides the actual implementation is based on the IRIS Performer [10] graphical render library. The specific Performer renderer links the visual scene graph to *Lightning* objects and reports changes of the geometry of a visual object to the underlying library and, as a result, to the graphics device hardware.

3.2 Event propagation

Objects within the pool and their links form a directed graph (link graph) which is evaluated at each simulation step. Starting with output from sensor objects, information is propagated along links. The behavior description code of a behavior object node is re-executed as soon as all nodes prior to this node (along the path of links) produced new output or at least had a chance to do so. Finally, if all graphs are evaluated, the procedure repeats. Output is defined by objects like for example cameras, which are part of the object pool, but are sampled asynchronously by the various renderer.

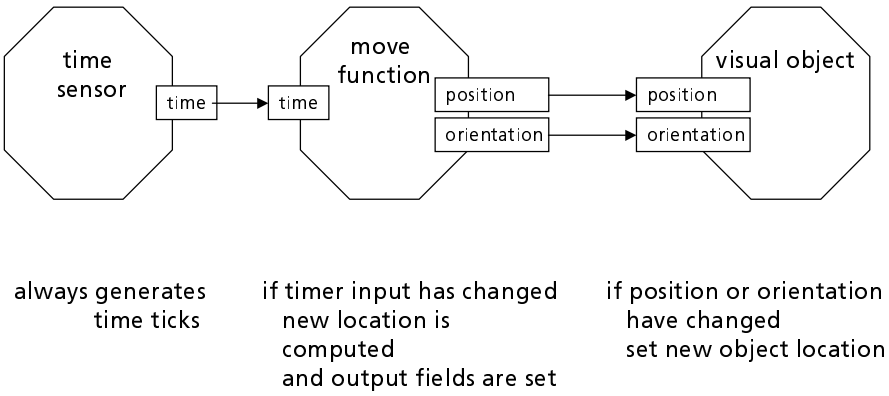

Fig 3. Example of event propagation

3.3 Multilanguage/multiparadigm programming

Lightning does not rely on a specific language but supports a heterogeneous, mixed-language approach. That is, behavior objects written in, for instance, C++ can be freely replaced by objects written in an interpreted language and vice versa, even at runtime. So far, we have used the Tcl programming language [8] as it is easy to integrate into other systems and as a free software package many resources and extensions. With its self evaluating property, the "eval" command it also has an expressive power similar to the "lambda" of Lisp or other AI languages. This enables a compact description of high-level behavior.

3.4 Multiprocessing

Multiprocessing is provided on the device level. All devices, which have significant differences in the update-rate are scheduled in another process. This holds basically for all input devices. Conceptually all render modules have their own processes. The IRIS Performer render library provides its own extended multiprocess scheme. The *Application Control Module* can distribute independent subgraphs to different processes. Often a link graph can be divided into two or more subgraphs, which are not connected through links.

3.5 Superimposing virtual space and user space

One basic approach to designing a projection model is to achieve an exact match of virtual and physical space. That is, the apparent size of an object should be independent of the projection system. If this is the case, the handle of a teapot would have exactly the same size if seen on a CRT screen as if seen on a wall projection. Consequently, the user can interact naturally with the teapot without the need for scaling the

geometry model. In *Lightning* the projection system is specified simply by providing a geometric model (with exact physical dimensions) of the projection device. With this data, the projection relations are automatically configured by the system.

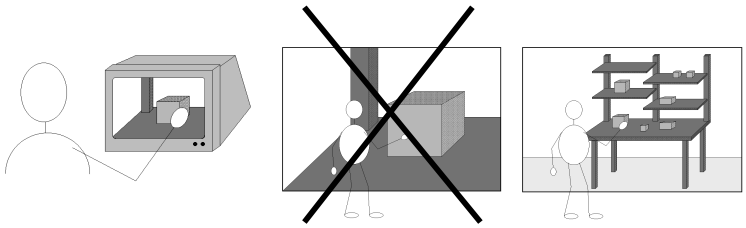

Fig 4. Projection system independence of object size

Location measurement calibrations are actually also an important factor for exact spatial matching. In *Lightning* this feature is provided on device module level.

3.6 Synchronicity

Modern virtual reality environments often consist of various modules, each operating with different update rates. That is, they have different virtual time bases with different granularity. It is necessary to synchronize these modules to obtain consistent or causal system behavior. The internal use of virtual time by different parts of the system is completely transparent. An obvious case is the synchronization of rendering: The sound of a ping pong ball should be delivered by the acoustic renderer exactly in that moment when the user perceives the visual sensation of the ball hitting the paddle. The *Lightning* system is designed to use a mechanism to transform discrete states into a continuous representation that is capable of serving an object state at a definite virtual time either via interpolation or extrapolation. This mechanism links different simulation modules to allow translation of different virtual time bases. The internal structure of the interpolation/extrapolation is considered as a basic system service and therefore it remains hidden for the application developer.

3.7 Extensibility

An important key feature of *Lightning* is its extensibility. All application objects are accessible via the Tcl interface. Behavior scripts can be developed to define the functionality of applications. New applications can be built completely on this layer.

On the core system layer we use a combination of object oriented techniques and operating system features for uncoupling the modules. Application objects inherit the interface and communication properties from a base class. The communication process operates only on base class features. The configuration and initialization communication is strictly string based to enable run time coupling.

All *Lightning* system libraries are so-called shared objects, which are linked at runtime. This feature is used primarily for ease of maintenance. Application objects are by default shared objects and can be accessed immediately by the Tcl interface without coding or recompiling. This is a major advantage for the extensibility of the system. Application objects can easily be developed; only common system interfaces have to be included. The extension on C++ level is also independent of static linking with the system.

4.0 Related Work

This section reviews related work on VR system architecture. There have been a lot of proposals for augmenting visual output in VR with other media, especially for spatial audio output. But, often caused by hardware performance limitations, very few of them provide a common framework which uniformly integrates all media currently available.
In VPL's Body Electric [2], users specify relations between virtual world entities and I/O devices in a dataflow diagram editor. This dataflow approach can also be found in a variety of similar systems such as SGI's Open Inventor$^{(TM)}$ [12]and VRML 2.0 [11]. A pure dataflow approach, however, came out to insufficiently support program modularity. Hence, VRML allows for defining complex object behavior within so-called script nodes, often written in the Java programming language. As mentioned earlier, the VRML model is very similar to our behavioral semantics system, particularly the object pool. But the specification of VRML is not without ambiguity, so behavior tends to depend on the actual implementation. Aimed mostly at internet applications, existing VRML browser, by contrast, do not provide ways for flexibly accessing modern output systems such as force feedback or wall projection with tracked shutter glasses.

Many commercially available products such as Sense 8's WorldToolKit (WTK) or Division's dVS system provide good support for most visual and acoustic output configurations, but do not include force output. They usually provide programming interfaces for C++ and often specifically designed interpreted languages. Other researchers provide additional languages such as Python [9] OML [5], or, for high-level interaction, Scheme [3]. Aimed at a broad range of application domains, *Lightning* could not rely on a single language but had to provide means to support many existing paradigms. The Alice system [9] also automatically separates simulation and rendering

into different processes. *Lightning* has adopted this approach, but provides a finer granularity as both simulation and rendering processes are automatically divided into subprocesses if more processors are available. The Avocado system of GMD [3] follows an approach similar to our system. It is based on a concept called ‘Performer with fields’ but is closer dedicated to the IRIS performer graphics library as *Lightning*.

5.0 Conclusion and future work

In this paper we have shown how recent requirements affect the architecture of VR systems. They need to take into account various i/o media and to provide ways for defining interactions and behavior and synchronize all these modules. The VR system *Lightning* has been implemented with these design issues in mind. It integrates audio and video output and is open for other media. Its multi-language behavior specification allows for more flexible and faster behavior prototyping. *Lightning* has shown its usefulness at various occasions in real-world applications ranging from engineering to entertainment.

Having implemented some of the design considerations the next step is to evaluate the performance. Next steps include research of the physiological and cognitive aspects of the perception of time, to be able to adapt system behavior better to the user. Furthermore the system should behave dynamically such that objects have weight and elasticity because it seems to be considerable useful to help user to interact more intuitively.

6.0 References

[1] Astheimer, P., Dai, F., Göbel, M., Kruse, R., Müller, S., Realism in Virtual Reality, Artificial Life and Virtual Reality, ed. by N. Magenat-Thalman and D. Thalman, 1994.

[2] Adachi, Y., Kumano, T., Ogino, K., Intermediate Representation for Stiff Virtual Ob-jects, Proc. IEEE VRAIS, Research Triangle Park, N. Carolina 1995.

[3] Hasenbrink F., Avocado system, Unpublished White Paper, GMD Department Visualisation and Media Systems Design, (Bonn St. Augustin 1997), (see http://viswiz.gmd.de/?hase/Avocado.html)

[4] Blanchard, C., Burgess, S., Harvill, Y., Lanier, J., Lasko, A., Obermann, M., Teitel, M., Reality built for two: A virtual reality tool, Proc. 1990, Symp. on Interactive 3D Graphics, Snow Bird, (Utah 1990).

[5] Green, M., and Halliday, S., A Geometric Modeling and Animation System for Vitual Reality, Communications of the ACM, Vol. 39, No. 5, (May 1996).

[6] Landauer J., Blach R., Bues M., Rösch A., Simon A.: Towards Next Generation Virtual Reality Systems, Proc. IEEE Conf. Mulitmedia Computing & System, (Ottawa 1997)

[7] Mark, W. R., Randolph, S. C., Finch, M., Van Verth, J. M., and Taylor, R. M., Adding Force Feedback to Graphics Systems: Issues and Solutions, Proc. ACM SIGGRAPH 96, (New Orleans 1996).

[8] Ousterhout, J., Tcl and the Tk Toolkit (Addison-Wesley, Reading, Massachusetts 1993).

[9] Pausch, R et al., A Brief Architectural Overview of Alice, a Rapid Prototyping System for Virtual Reality, IEEE Computer Graphics and Applications, (May 1995).

[10] Rohlfs, J., Helman, J., IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics, Proc. ACM SIGGRAPH, (Orlando 1994).

[11] The Virtual Reality Modeling Language (VRML) Specification Version 2.0, ISO/ IEC CD 14772, (see http://vrml.sgi.com/moving-worlds/spec/ index.html)

[12] The Open Inventor C++ Reference Manual, (Addison-Wesley, Reading MA 1995)

[13] Schraft et al.: A Fuzzy Controlled Rendering System for Virtual Reality Systems Optimized by Genetic Algorithms, Proc. 2nd Eurographics Workshop on Virtual Environments, (Springer, Wien, 1995)

[14] Azuma R, Bishop G.: A Frequency-Domain Analysis of Head-Motion Prediction, Proc. ACM SIGGRAPH 95, (Los Angeles 1995).