

# Lightweight Programming for VR : Toward a Persistent Virtual Laboratory

Luc RENAMBOT<sup>1</sup>      Henri E. BAL<sup>1,2</sup>      Desmond GERMANS<sup>2</sup>  
Hans J.W. SPOELDER<sup>2</sup>

<sup>1</sup>Division of Mathematics and Computer Science

<sup>2</sup>Division of Physics and Astronomy

Faculty of Sciences, Vrije Universiteit

De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

## 1 Introduction

In recent years, the availability of virtual reality, high-performance computing, and high-speed networks has opened opportunities for completely new types of applications. Traditionally, scientific visualization displays computations in an off-line and non-interactive fashion. Scientific visualization is moving away from off-line and batch-oriented management to a visual style of processing, using collaborative and interactive virtual reality methods and computational steering [5]. To become widely used by scientists, virtual reality environments should provide tools to connect, visualize and control on-going simulations. Furthermore, a scientist wants to learn from his virtual reality experience. To facilitate this, the environments should provide means to interactively identify and quantify features of the simulated data from the visual domain [15]; In short, scientists should be able to *measure* the data, using a variety of measurement paradigms.

Existing methods for measuring in virtual reality are based on ideas like probing the visual domain, deriving quantities from the simulation prior to visualization or altering the simulation to generate required data directly. However, these methods are too restrictive, rely on advanced programming skills of the scientist, or are batch-oriented by nature.

To cope with these problems, we present a new programming environment that enables scientists to easily and interactively steer an un-modified simulation from a virtual reality environment. Rapid prototyping of interactive environments will bring more scientists to use VR environments. A key point is to close the loop between simulation, immersive visualization, interaction in simulation domain and steering, giving the scientist deeper insight into the simulated phenomenon. A cy-

cle we refer to as virtual measuring [6] or high-level steering.

Using a high-level XML description of the simulation, a relational database, and several Python modules, the system provides a communication layer to control the execution of the simulation, an easy to program measuring framework in VR, and a technique to manage scientific datasets. We describe a preliminary case-study dealing with diode laser simulation.

The main contributions of our work are as follows :

- a high-level steering system in a scripting language,
- an easy to program measurement framework in VR,
- a technique to start, browse, and replay scientific simulations,
- a case study on theoretical physics, so called *Sisyphus attractor*.

This paper is structured as follows. In Section 2, we survey related research. In Section 3, we present our environment. Using an real program, we show the coupling of a simulation to a virtual environment in Section 4. Finally, we present some conclusions.

## 2 Related work

So far, bringing non-VR scientists to exploit VR and tele-immersion system is very cumbersome process. It requires a great skill of programming either in VR for the scientist or a very good understanding of the simulation for VR specialist. Usually, the user of the simulation is unable to port the program to a VR system for an interactive steering session.

A great deal of research has been done on the so called “Grid Computing” research area [5], providing languages, tools, and environments to create new applications that were not conceivable before. For instance, one can cite world-wide collaboration in virtual reality, or real-time data-mining of large data sets. Usually, this becomes possible at the expense of high programming complexity.

On another side, computational steering focuses on the interactive control of a simulation during its execution [11]. The scientist can control a set of parameters of the program and react to the current results. Computational steering enhances the productivity of the scientist by giving a problem-solving environment. The idea of a *virtual laboratory*, where scientist could analyze their datasets as being produced, is explored in some research projects. Among them are VIDL [9], Distributed Laboratories [12, 13], or Cactus [1], the latter trying to provide a collaborative and problem-solving environment for large-scale simulations. However, existing systems are designed to build new applications or require modifications to the source code of the application, and thus are unsuitable if the source code is unavailable.

AccessGrid technologies [16] as a new medium for group collaboration in virtual meeting rooms, using real-time audio, video, and distributed presentation provide a new way for tele-conference and group work, but still lack support for scientific data analysis.

No environment so far provides the functionalities needed for an immersive steering and measuring environment which is easy to use by non-expert, highly dynamic to deal with collaborative aspects, and reconfigurable for a rapid prototyping approach. The steering system *CAVEStudy* [14] and the high-level VR toolkit VIRPI [7] are first steps in the direction, where the user is able to efficiently explore his simulation in a VR world, and to get a deeper insight of the studied phenomena. But, it lacks flexibility for several reasons. First, the file format for the simulation description is not standard. Second, the code generator produces C++ classes that have to be compiled and linked into a VR framework. Finally, the steering system and the VR framework are still two distinct components that have to be linked by the programmer.

However, new programming paradigms and tools have been lately designed for flexible, distributed, and efficient computing, mainly in the context of Internet programming. Among them are scripting languages, such as Perl or Python, automatic library wrapper for interpreted languages, such as SWIG, lightweight efficient database, such as MySQL, extensible file format, such as XML. These tools are usually overlooked by

scientific programmers who prefer to develop their own software in well-known compiled languages (C/C++). However, this software proves to be efficient and flexible [3]. It provides a high-level interface, while underneath it maintains efficient execution in layers such as graphics rendering and network communication.

### 3 Toward a Virtual Laboratory

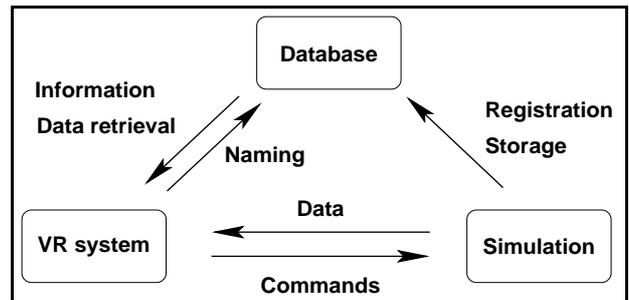


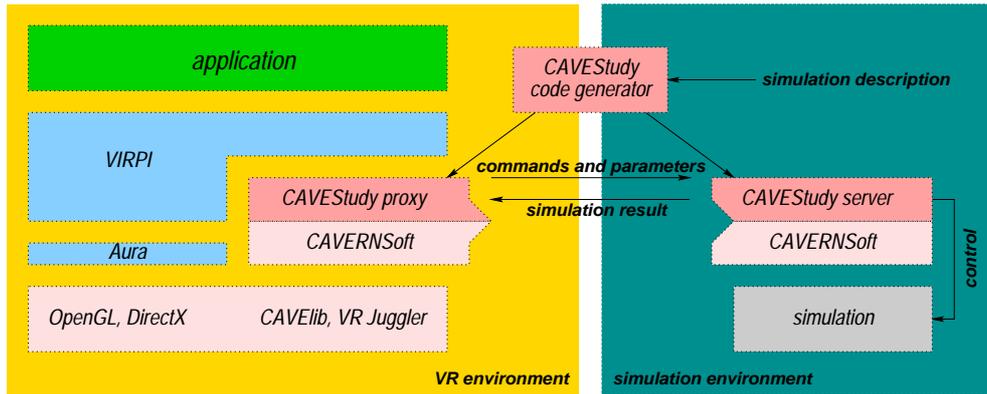
Figure 1. Basic components of the system

The goal is to get the scientist to bring himself his simulation program to any VR system (workstation, Cave, Immersadesk, Wall, ...) and let him explore, analyze, measure, and control the data produced. In short, he should be able to produce data, to store the results, and to learn something from his interactive visualization session. We see several components in such an environment (figure 1) : a VR system (generic term for immersive or semi-immersive setup) connected to computing resources where the simulation runs and storage resources to keep the datasets produced. Between the components, informations are exchanged to complete the tasks : such as naming information (where the simulation is running), data discovery (what new dataset has been produced), and steering information (command to the simulation, data retrieval from the simulation).

In this paper, we explore some possibilities to interconnect these components.

#### 3.1 Computational Steering in VR

*CAVEStudy* [14] is a system that allows the scientist to steer a program from a virtual reality system without requiring any modification to the program. It enables an interactive and immersive analysis of simulations running on remote computers. *CAVEStudy* allows non-experts in VR to couple their simulation to virtual environments. It uses the CAVERNSoft [10]



**Figure 2. The software layers**

toolkit as network layer, which is designed to provide a high performance network infrastructure for tele-immersion and collaboration.

The scientist models the simulation as a set of input, output, and graphical objects in a XML syntax. These objects are the input parameters of the simulation and the data produced. Given such a description, our system generates a wrapper around the simulation to control its execution on a remote computing system, which usually is a supercomputer or a cluster. The data produced by the simulation is then packed and sent to the computer hosting the virtual environment. A proxy for the remote simulation is built to receive the data generated by the simulation. This proxy is plugged into a virtual reality environment, where it updates the objects described by the scientist. CAVERNSoft provides the communication and persistence layer needed by our infrastructure. Our infrastructure consists of an XML description syntax, a code generator, and a virtual reality environment. Thus, it is possible to visualize and control the program directly in the domain space of the simulation.

For greater ease of use, *CAVEStudy* is integrated into our VR software environment, as describe in figure 2. The left side shows the VR environment build with the VIRPI toolkit [7], and the various layers that are running there. The right side shows the simulation environment, and the connection that is made with *CAVEStudy*. VIRPI is an integral toolkit for interactive visualization in virtual reality environments. It defines a framework to build applications that allow the user to interact with simulation data and describe virtual measurement tools for the visualized data. It is aimed at non-VR experts. With very little programming effort, the programmer can create an interactive VR application that suits the needs of the apparent field of research. VIRPI hides platform differences, pro-

vides scene graph support, simple shapes, text, graph loading and access to VR hardware. Also, an easy to use C++ interface allows the programmer to define interactive behavior for measurements, object examination and the selection of subspaces.

### 3.2 XML as a glue

As mentioned earlier, we use the XML file format to describe the data of a simulation. It concerns the input parameters of the simulation but also the result of a run. From this file, we generate modules able to control a simulation execution. XML in such a mode is used a protocol description between the software components. Figure 3 shows how we use XML information stored in the database to connect several software components in unified way. This way of using XML could be related to existing works like the CCATT project [4] where XML-schemas (offering a more strict semantic than regular XML-DTD) to connect distributed software component. One can cite also work at the CACR project at Caltech where XSIL is designed as an XML extension to manage scientific data in XML, or the XDF project from NASA focusing on data management with XML for astronomical observation.

### 3.3 Scripting

*SWIG* [2] is a tool for integrating scripting languages like Perl or Python with C and C++ libraries. Python is an object-oriented scripting (or interpreted) language widely popular in particular in Internet programming. It offers a large and portable set of modules (for string handling and process control to XML parsing and database management). It combines in a clean interface the C++ object-oriented aspect and the Perl power of expression. Using SWIG, we

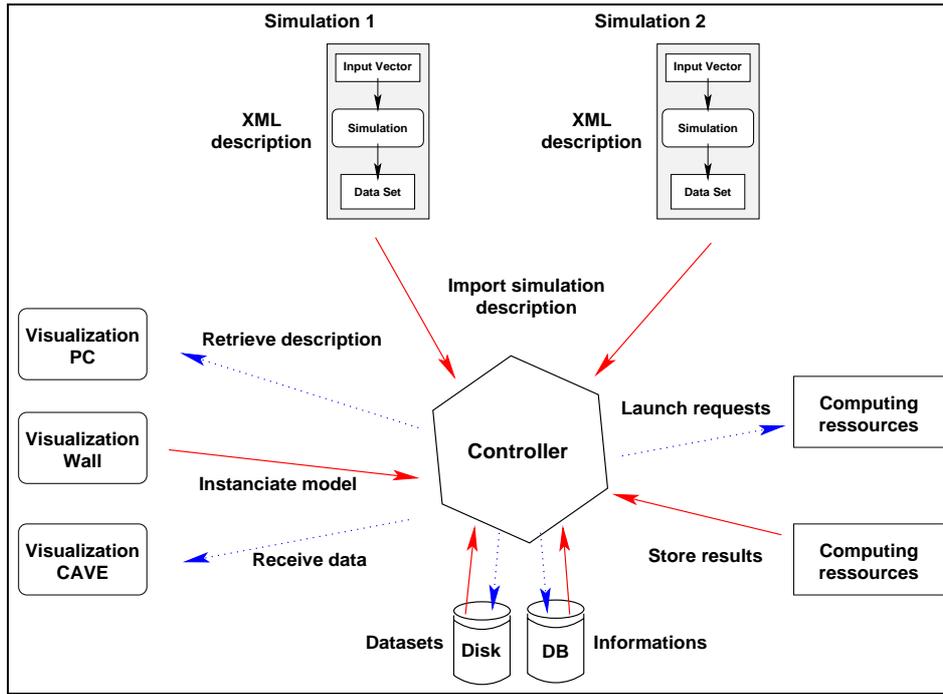


Figure 3. XML as a protocol description / as software component glue

produced a Python version of Aura, the low level graphic layer of our VR toolkit. The high-level interaction and widget layer of our toolkit, VIRPI, is easily translated entirely to Python. This way, the performance of Aura remains untouched, and extra flexibility is gained on a higher level. This mechanism gives a great ease of use, combining scripting language and graphic efficiency. In the same way, we wrapped the low-level *CAVEStudy* classes into a Python module, giving us a flexible access o CAVERNSoft functionalities (on which *CAVEStudy* is based).

Scripting and an explicit knowledge of the input parameters of a simulation allow us to write very easily a large variety of applications, interactive or not, to exploit different aspects of hardware setup (from the office workstation to the full immersive system or a large wall display). For instance, one could write non-VR application such as a minimal batch system to explore extensively the input parameter space of a simulation.

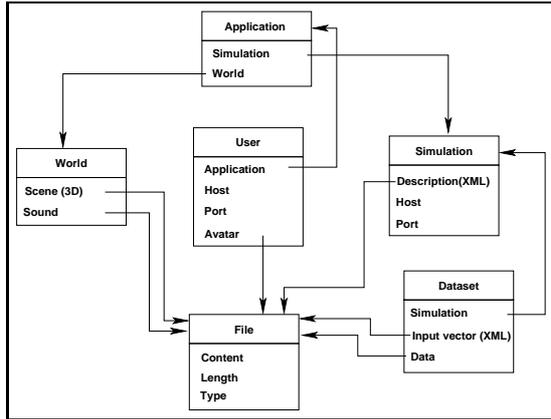
### 3.4 Database

The use of an relational database in an interactive visualization system could seem awkward at first. We propose to use it as a persistent access point to the virtual laboratory. it manages the persistence of the

datasets produced and the naming functionality. In our project, we use MySQL, a free, fast, and widely used database system with an SQL front-end. It offers plenty of interesting functionalities such as : networked repository (multiple users from multiple site can access the data simultaneously), fast response time using multi-threaded daemons, some security features, data can be easily replicated over multiple sites, and several languages bindings (C, Perl, Python, C++) which are easy to use.

We can see two modes of interaction with a simulation, in respect to the database. First, a direct mode (described in [14]) with live coupling between the simulation and the VR environment. Second, the system could be used as data repository where the user can browse previously produced datasets or start new simulation to produce data in a non-interactive mode. The same interface is shown to the VR environment in both modes, in such a way that one can switch between live data and existing datasets.

The current design of our database is sketched in figure 4. The data stored in the database are simulation informations, user informations, and datasets already produced. We wrote in Python several modules to store and retrieve information : registration as a new user in the system (login), creation of a new simulation entry, registration of a new running



**Figure 4. Element of the database**

simulation, upload and download of a dataset. This system allows us to easily answer questions like “where a given simulation is running”, “where are the users connected to any simulations”, or “what dataset are available and where are they stored”. One key point is that the XML description of a simulation is also stored in the system, and is used by the runtime for the communication between the components. Datasets can be store within the database if they are small and for ease of use. A more general way is to store the name of an URL pointing to the data file (<http://...>, <ftp://...>) accessible transparently. The data are stored within that file in a compact way (for example in binary mode using the marshaling/demarshaling facilities of Python).

A session at the simulation-side consists of several steps : Registration of name, host, and communication port in the simulation table; Upload of the XML file describing the simulation; Generation a Python wrapper for the simulation using the runtime XML parser; Instantiation of that wrapper and waiting for commands; Send the data updates if there’s one (or several) live users; Store the results in the database at the end of the run. A session at the VR side-could be seen as : Register as a user to the database; Download the required information to connect to a running simulation such as host, port, XML description; Exploration of a pre-computed dataset by retrieving the location of the data and the XML description (Both approaches, live simulation or precomputed data, should be almost transparent : it consists of the instantiation of an input vector for the selected simulation); Modification of the input parameters (meaning interactive steering by sending commands to the simulation in a live session, or switching to a new dataset for a data mining session); Finally,

measuring in data space using VIRPI to get a better insight of the phenomenon under study.

### 3.5 Putting It All Together

We think our approach provides several advantages over traditional approaches. It removes of the distinction between interactive steering, parameter-space exploration, batch scheduling, or VR exploration of a dataset. All data produced are stored by default. The scientist can still delete irrelevant dataset afterwards. Interactive session are not “result-less” : when the scientist leaves the VR setup (Cave, PowerWall), data are still available for further examination on a regular desktop. The knowledge of the input parameters and data produced in an XML description allows new ways to use a simulation program, such as systematic or intelligent exploration of the parameter space by specifying a range or a function for each parameter, or feature extraction over the data produced. Moreover, data can be produce in a over a long period without user interaction and be examined later on by writing a very simple script program. Our C++ APIs (CAVEStudy/VIRPI) are still available and fully operational with Python components. After a rapid prototyping in Python and for demanding applications (large dataset for instance), a C++ implementation could improve the performances.

*The full paper gives some program and XML samples to illustrate the description.*

## 4 Application

To evaluate our scripting approach of coupling a simulation and a virtual-reality environment, we implemented an application from the theoretical-physics field dealing with the chaotic behavior of diode laser in some given conditions. We address in this example the ease of use, the usability of such a method, and the added value for the user.

We implemented the visualization of a diode laser behavior, referred to as the *Sisyphus Attractor* [8]. Numerical simulations are performed for a semiconductor diode laser, subject to optical feedback. Due to the feedback, the resulting dynamical system has infinite degrees of freedom. The exploration and investigation of such a large data set calls for the immersion of the user into a representation of the parameter space. A simulation run generates a trajectory in such a space. Directly in simulation space, the user is able to tune four parameters of the simulation to explore their inter-relations. Both interactive sessions to explore

a limited set of values and extensive off-line dataset generation are useful for this application, which make it a perfect case study for our flexible environment.

*This part describes in length the application and the added value of our system in the full paper.*

## 5 Conclusion

We sketched a new environment for steering from within virtual reality environments in a flexible way. Both Python and C++ bindings are provided to the *CAVEStudy* and *VIRPI* toolkits. The system provides interactive immersive analysis and control of a simulation running on a remote computer. It does not alter the simulation program but interacts with the simulation input and output parameters described as XML attributes. Using modern and lightweight software components (Python modules, XML files, relational database), we draw some functionalities needed to ease and enhance the use of VR as a scientific visualization tool. As a case study, we applied our system to an existing unmodified simulation program from the field of theoretical physics. On that examples, we showed the added values of our system such as ease of use, flexibility, and in general a new way to manage scientific programs.

## References

- [1] G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, and E. Seidel. The Cactus Code: A Problem Solving Environment for the Grid. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, Aug. 2000.
- [2] D. M. Beazley. SWIG: an easy to use tool for integrating scripting languages with C and C++. In U. Association, editor, *4th Annual Tcl/Tk Workshop '96*, pages 129–139. USENIX, July 1996.
- [3] D. M. Beazley and P. S. Lomdahl. Lightweight computational steering of very large scale molecular dynamics simulations. In *Supercomputing '96 Conference Proceedings*. ACM Press and IEEE Computer Society Press, Nov. 1996.
- [4] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A Component Based Services Architecture for Building Distributed Applications. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, Aug. 2000.
- [5] A. Foster and C. Kesselman. *The Grid: Blueprint for a New Computer Infrastructure*. Morgan Kaufman, 1998.
- [6] D. Germans, H. J. Spoelder, L. Renambot, and H. E. Bal. High-Level Steering : Measuring in Virtual Reality Environments. In *Proceedings of the ASCI, The Netherlands, 2001*, 2001.
- [7] D. Germans, H. J. Spoelder, L. Renambot, and H. E. Bal. VIRPI: A High-Level Toolkit for Interactive Scientific Visualization in Virtual Reality. In *Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop, Stuttgart, Germany, 2001*, 2001.
- [8] C. Mirasso, M. Mulder, H. Spoelder, and D. Lenstra. Visualization of the Sisyphus Attractor. *Computers in Physics*, 11(3):282–286, May/June 1997.
- [9] U. Obeysekare, F. Grinstein, and G. Patnaik. The Visual Interactive Desktop Laboratory. *IEEE Computational Science and Engineering*, 4(1):63–71, Jan. 97.
- [10] K. S. Park, Y. J. Cho, N. K. Krishnaprasad, C. Scharver, M. J. Lewis, J. Leigh, and A. E. Johnson. CAVERNsoft G2: A toolkit for high performance tele-immersive collaboration. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 8–15, Oct. 2000.
- [11] S. Parker, M. Miller, C. Hansen, and C. Johnson. An Integrated Problem Solving Environment: the SCIRun Computational Steering System. In *Hawaii International Conference of System Sciences*, pages 147–156, Jan. 1998.
- [12] B. Plale, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter. From Interactive Applications to Distributed Laboratories. *IEEE Concurrency*, pages 78–90, April-June 1998.
- [13] D. Reed, Giles, and C. Catlett. Distributed Data and Immersive Collaboration. *Communication of the ACM*, 40(11), Nov. 1997.
- [14] L. Renambot, H. E. Bal, D. Germans, and H. J. Spoelder. CAVEStudy: an Infrastructure for Computational Steering in Virtual Reality Environments. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 57–61, Aug. 2000.
- [15] H. J. Spoelder. Virtual Instrumentation and Virtual Environments. *IEEE Instrumentation and Measurement Magazine*, 3(3):14–19, 1998.
- [16] R. Stevens. ActiveSpaces: The access grid, active mural and advanced visualization systems. In D. Ebert, M. Gross, and B. Hamann, editors, *Proceedings of the 1999 IEEE Conference on Visualization*, pages 16–18, Oct. 1999.