

Lightweight Programming for VR: Towards a Persistent Virtual Laboratory

Luc RENAMBOT¹ Henri E. BAL^{1,2} Desmond GERMAN²
Hans J.W. SPOELDER²

¹Division of Mathematics and Computer Science

²Division of Physics and Astronomy

Faculty of Sciences, Vrije Universiteit

De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

Abstract

In this paper, we sketch a new environment for steering from within virtual reality environments in a flexible way. The system provides interactive immersive analysis and control of any simulation program. It does not alter the simulation program but interacts with the simulation input and output parameters. Python modules, XML files, and a relational database ease and enhance the use of VR as a scientific visualization tool. We demonstrate the use of our system to a scientific simulation. For the non-VR expert, the added values of our system are the ease of use, the flexibility, and a new way to manage scientific programs.

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 [6]. To become widely used by scientists, virtual reality environments should provide tools to connect, visualize and control on-going simulations. Furthermore, a virtual reality session must be a learning experience for the scientist. To facilitate this, the environments should provide means to interactively identify and quantify features of the simulated data in the visual domain[18]. 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 unmodified 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 the simulation domain and steering, giving the scientist deeper insight into the simulated phenomenon. We refer to this cycle as virtual measuring [16] or high-level steering. Another important aspect is the capability to manage the datasets stored into a persistent space from which they can be retrieved for analysis. Moreover, the produced datasets should be globally available, i.e. from the supercomputer where they are produced to the visualization system and from the office workstation.

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.

The main contributions of our work are as follows:

- a high-level VR steering system in a scripting language,
- a technique to start, store, browse, and replay

scientific simulations,

- a preliminary case study using a theoretical physics simulation program.

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

2 Related work

Much research has been done on the so-called “Grid Computing” research area [6], resulting in 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.

Computational steering focuses on the interactive control of a simulation during its execution [12]. 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 scientists analyze their datasets while they are produced, is explored in some research projects. Among them are VIDL [10], Distributed Laboratories [13, 14], and Cactus [1], the latter trying to provide a collaborative 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 [19] provides a new medium for group collaboration in virtual meeting rooms, using real-time audio, video, and distributed presentation. Nevertheless, this new way for tele-conference and group work still lacks support for scientific data analysis.

No environment so far provides the functionalities needed for immersive steering and measuring which is easy to use by non-experts. Moreover, such an environment should be highly dynamic to deal with collaborative aspects, and reconfigurable for a rapid prototyping approach. The steering system *CAVEStudy*[15] and the high-level VR toolkit *VIRPI*[7] are some first steps in this direction, where the user is able to efficiently explore a simulation in a VR world, and to get a deeper insight of the studied phenomena. However, the previous implementation of *CAVEStudy* 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 the VR framework. This steps introduce extra manipulations for the user and incompatibilities in a heterogeneous environment. An interpreted approach would simplify the manipulation of programs and would increase portability for the end user. Finally, the steering system and the VR framework are still two distinct components that have to be linked by the programmer.

We address this problem of lack of flexibility using several tools that have been designed for flexible, distributed, and efficient computing, mainly in the context of Internet programming. Among them are scripting languages, such as Perl and Python, automatic library wrappers for interpreted languages, such as SWIG, lightweight efficient databases, such as MySQL, and extensible file formats, 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, they proved 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 Towards a Virtual Laboratory

So far, bringing non-VR scientists to exploit VR and tele-immersion systems is a 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 the VR specialist. The user of the simulation is often unable to port the program to a VR system for an interactive steering session.

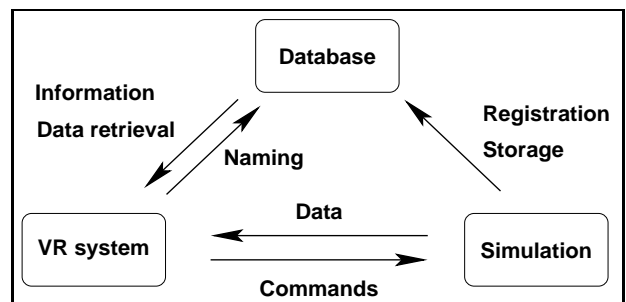


Figure 1. Basic components of the system

For a wide acceptance of such a system, the scientist should be able to connect the simulation program to any VR system (workstation, Cave, Immersadesk, Wall, etc.) without the help of a VR expert. It would

be a simple tool to explore, analyze, measure, and control the data produced.

In short, the user should be able to produce data, to store the results, and to learn something from the 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, information is 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).

3.1 Computational Steering in VR

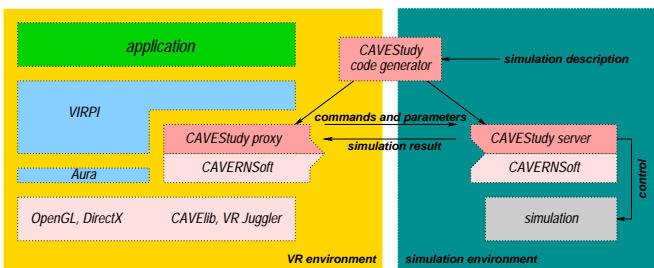


Figure 2. The software layers

CAVEStudy [15] is a system that allows the scientist to steer a program from a virtual reality system without requiring any modification to the application 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* [11] 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 an XML syntax. These objects are the input parameters of the simulation and the produced data. 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 built with the *VIRPI* toolkit [7], and the various layers that are running there. One important aspect is that using the *Aura* package, we abstract the different low-level capabilities among VR systems. The right side shows the simulation environment, and the connection that is made with *CAVEStudy*. *VIRPI* is an integrated 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. *VIRPI* 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 given field of research. *VIRPI* hides platform differences, provides 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

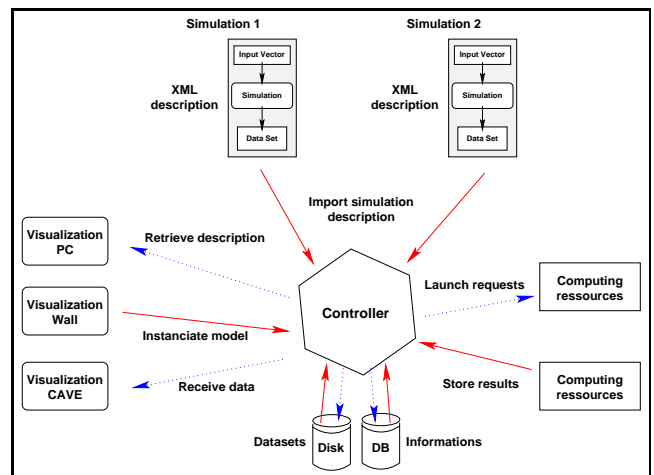


Figure 3. XML as a software component glue

As mentioned earlier, we use the XML file format to

describe the data of a simulation including the input parameters and the result of a run. From this file, we generate modules able to control a simulation execution. XML is used as 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 an unified way. This way of using XML could be related to existing works like the CCATT project [5] where XML-schemas (offering more strict semantics than regular XML-DTD) to connect distributed software component. One can cite also work such as XSIL [4] which design an XML extension to manage scientific data in XML, or the XDF project [17] 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 (from string handling and process control to XML parsing and database management). It combines the object-oriented aspects of C++ and the power of expression of Perl in a clean interface. Using *SWIG*, we 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 great ease of use, combining a scripting language with graphic efficiency. The same technique is applied to *CAVERNSoft*[8] (on which *CAVEStudy* is based) giving us access to advanced networked functionalities such as RPC, remote file access, and high-bandwidth communication from simple scripts.

Scripting and an explicit knowledge of the input parameters of a simulation allow us to very easily write 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 optimization program to explore the input parameter space of a simulation extensively. Figure 4 shows such a program in pseudo-language searching the optimal set of input values maximizing the outcome of the simulation. Such programs could also populate a database, storing the

results of a large number of runs for later exploration in VR.

```

1  # Simulation taking input parameter I and J
2  max = -Inf
3  for I in range(Imin,Imax,Istep)
4      for J in range(Jmin,Jmax,Jstep)
5          result(I,J) = run simulation(I,J)
6          if result(I,J) > max :
7              max = result(I,J)
8              store result(I,J) in the database
9      end
10 end

```

Figure 4. A scripting example

3.4 Database

The use of a 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 sites can access the data simultaneously), fast response time using multi-threaded daemons, some security features, easy replication of data 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, there is a direct mode (described in [15]) 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 simulations 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 5. The data stored in the database is information about the simulation, the user, as well as previously produced datasets. Using Python, we wrote 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 simulation, upload and download of a dataset. This system allows us to easily answer questions like “Where is a given simulation running”, “Where are the users connected to any simulation”, or “What

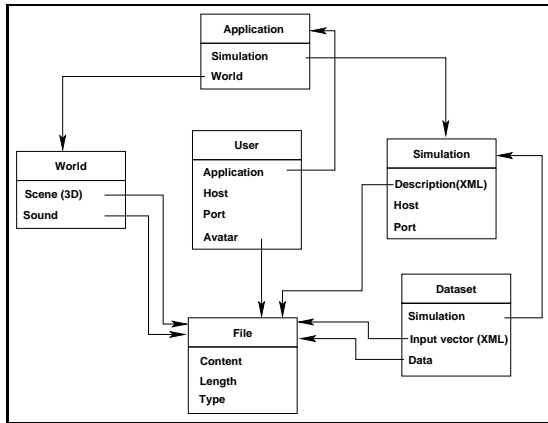


Figure 5. Element of the database

datasets 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 stored within the database 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/demmarshaling facilities of Python).

A session at the simulation-side consists of several steps: Registration of name, host, and communication port in the simulation table; Uploading of the XML file describing the simulation; Generation of a Python wrapper for the simulation using the runtime XML parser; Instantiation of that wrapper and waiting for commands; Sending of the data updates if there are live users; Storing of 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
- Downloading of the required information to connect to a running simulation such as host, port, XML description
- Exploration of a precomputed dataset by retrieving the location of the data and the XML description. Both approaches, live simulation or precomputed data, should be 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 dataspace using VIRPI gives a better insight into the phenomenon under study.

3.5 Putting It All Together

We think our approach provides several advantages over traditional approaches. It removes the distinction between interactive steering, parameter-space exploration, batch scheduling, or VR exploration of a dataset. All data produced is stored by default. The scientist can still delete irrelevant dataset afterwards. Interactive sessions are not “result-less”: when the scientist leaves the VR setup, 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 produced over a long period without user interaction and be examined later on by writing a simple script program. Our C++ APIs (CAVEStudy/VIRPI) are still available and fully operational with Python components. After rapid prototyping in Python and for demanding applications (large dataset for instance), a C++ implementation could improve the performance.

4 Application

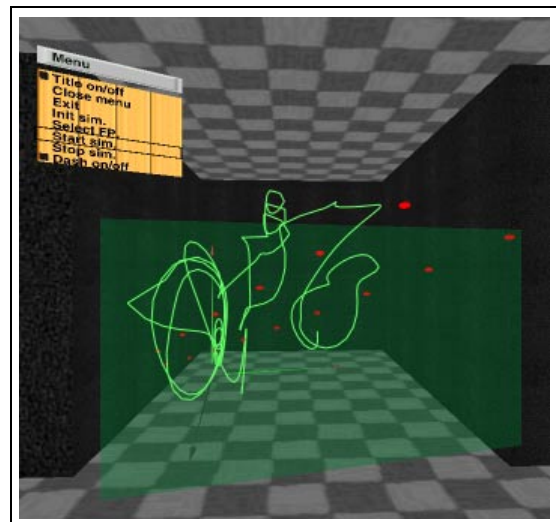


Figure 6. Diode laser simulation

To evaluate our scripting approach of coupling a simulation and a virtual-reality environment, we implemented an application from the domain of theoretical physics, the *Sisyphus Attractor* [9]. The application deals with the chaotic behavior of diode laser under given lab conditions. In this example, we address the ease of use, the usability of such a method, and the added value for the user.

```

1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <!DOCTYPE CAVEStudy SYSTEM "http://www.cs.vu.nl/
3      ~renabot/vr/cavestudy.dtd">
4  <CAVEStudy id="Sisyphus">
5
6      <!-- New data type      -->
7  <struct id="vector4D">
8      <field id="v" type="vector3D"/>
9      <field id="t" type="float"/>
10 </struct>
11
12 <!-- Simulation program -->
13 <simulation id="filsim">
14 <directory value="sim-sis"/>
15 <executable value="lkint"/>
16 <processes value="1"/>
17 <in type="commandline" default="null"/>
18 <out type="file" default="data.fil"/>
19 </simulation>
20
21 <!-- Input parameters      -->
22 <input id="alpha"          type="float" value="5.0"/>
23 <input id="pump_current"   type="float" value="140254.6"/>
24 <input id="feedback_rate"  type="float" value="0.00782"/>
25 <input id="starting_fp"    type="int"   value="0"/>
26
27 <!-- output data          -->
28 <output id="fixed_point"   type="vector3D" dim="1"/>
29 <output id="new_point"     type="vector4D" dim="0"/>
30
31 </CAVEStudy>

```

Figure 7. XML description of the diode laser simulation

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. A simulation run generates a trajectory describing the laser state in its parameter space. In the 3D space provided by *CAVEStudy*, we decided to focus on the most natural phase space from the physical point of view (the output power, the inversion, and the phase difference) as shown in Figure 6. The exploration and investigation of such a large data set calls for the immersion of the user into a representation of the parameter space. Directly in simulation space, the user is able to tune four parameters 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. A previous study [9] on the visualization of this simulation already gave a better insight into the dynamical behavior of the laser, but suffered severely from lack of interaction. With *CAVEStudy*, we linked the simulation running on an IBM SP2 to our CAVE. In a first step, the simulation computes some fixed points in the phase space for a given set of parameters. The user can interactively set the values of selected parameters using sliders. The fixed points serve as starting point of the simulation. These points are visualized, and the scientist can directly select one of these points to start the simulation. The computed trajectory is sent incrementally to the CAVE. The trajectory is visualized and can be manipulated by the scientist. The simulation can be stopped and re-started using a new starting fixed-point or different parameter values.

CAVEStudy's benefits are many-fold in this case; it is easier to use than a previous approach (batch-processing and off-line visualization); the study of the initial-condition sensitivity of the laser is enhanced by the ability to modify the parameters of the simulation interactively; since our system does not require modifications of simulation code, we can deal very easy with the changes of a code still revised frequently; the interactive way in which physicists could test hypotheses and investigate the behavior of the diode laser helped them to gain a better insight in this complex system.

To bring this program to VR, the user has first to write an XML description of the simulation. In Figure 7, we can see that the user defined a new data type called *Vector4D* which holds a 3D coordinate and a time value. The input parameters (lines 19–22) are *alpha*, *pump_current*, *feedback_rate* which are floating-point value, and *starting_fp* which represents the initial condition of the system. The output of the program (lines 24–25) consists of a set of points stored in a parameter called *fixed_point*, and the current state of the system stored in *new_point*. This XML file is then feed into our runtime system, as shown in Figure 8, where a client is then generated (lines 11–14). Then, this client connects to the remote simulation (line 18) according to the data found in the central database. Finally, the value of the input parameters can be set (lines 21–22).

When the previous two steps are performed, the user can plug the generated client into a VR program (Figure 9). The incoming data from the simulation is automatically received through a set of callbacks (lines 21–24). Finally, commands can be sent to the simulation to control its execution from the VR world (lines 27–35). For each run of the simulation, all the data

```

1 # Connection to MySQL
2 sql = cDatabase()
3
4 # Get the xml file
5 description,host,port = sql.GetSimulation("Sisyphus")
6 # Download the XML description
7 sql.retrieveFile(description, "simul.xml", "XML")
8 # Build an XML parser
9 parser = XmlParser(CavestudyXML())
10 # Parse the downloaded XML file
11 parser.Parse("simul.xml")
12
13 # Generate the client Python-program
14 parser.Generate()
15 # Load the client generated module
16 module = __import__("Sisyphus")
17 # Create a client to the remote server
18 client = module.Create(host,port)
19
20 # Set an input parameter
21 client.data["alpha"] = 4.5
22 client.send_alpha()

```

Figure 8. Connection to a remote simulation

received are automatically stored in a compact form on disk. At the same time, an XML file (same syntax than the description file) containing the value of the input parameters is generated to describe this run. From this pair of files, it is possible to replay off-line the entire simulation. Also, an entry in the database (cf. Figure 5) is created to describe this run, with the simulation description, an dataset identifier and the location of the data files designated using URLs. All this setup providing a persistent data space is transparent to the user. The runtime also provides simple functions to retrieve a given dataset from the database, and to replay it as it was generated live. The next step would be to implement an automatic retrieval and replay if the input values given by the user match an existing dataset. Another example is, using the script in Figure 4, to generate a large number of simulation runs to explore a parameter space, and then to analyse the produced solutions in a VR environment. At anytime, the user can switch to on-line steering by starting a new instance of the simulation.

This VR application allows the scientist to determine the inter-relations of the input parameters leading to a chaotic state of the laser. The ability to start and stop the remote program, to change quickly the input parameters and the initial condition of the simulation increase the understanding of the underlying physical process. The next step would be, using VIRPI [7], to implement some measurement functions to quantify the study.

```

1 from VIRPI import *
2
3 class cApplication(cvApplication):
4
5     # Constructor
6     def __init__(self):
7         cvApplication.__init__(self)
8
9         # Build the menu list
10        menu = cvContextMenu(3,"VR menu")
11        menu.AddItem(0, cvMenuItem("Start simul", 1))
12        menu.AddItem(0, cvMenuItem("Stop simul", 2))
13        menu.AddItem(2, cvMenuItem("Quit", 3))
14        menu.Scaling(avector(0.05,0.05,0.05))
15        menu.Rotation(aquaternionHPR(0,-90,0))
16        AddView(menu)
17
18        # Simulation
19        client = simul.Create(host,port)
20
21    # Idle loop
22    def Idle(self):
23        client.process()
24
25    # Data callback
26    def update(self, sender):
27        # Copy the value from sender
28        val = sis.data[sender]
29
30    # Menu callback
31    def onCommand(self, command):
32        if command == 1:
33            client.initialisation()
34            client.start()
35        elif command == 2:
36            client.stop()
37        elif command == 3:
38            client.shutdown()
39            exit()
40
41    # Main program
42    aInitialize()
43    ve = cvEnvironment(800,600,32,"Sisphus",0,v_theme_Ibiza)
44    application = cApplication()
45    ve.Application(application)
46    ve.Run()
47    aShutdown()

```

Figure 9. VR Python-program

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 functionalities needed to ease and enhance the use of VR as a scientific visualization tool, such as scripting, interactive visualization, and persistent data space. As a case study, we applied our system to an existing unmodified simulation program from the field of theoretical physics. On that example, 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] K. Blackburn, A. Lazzarini, T. Prince, and R. Williams. XSIL: Extensible Scientific Interchange Language. *Lecture Notes in Computer Science*, 1593, 1999.
- [5] 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.
- [6] A. Foster and C. Kesselman. *The Grid: Blueprint for a New Computer Infrastructure*. Morgan Kaufman, 1998.
- [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*.
- [8] J. Leigh, O. Yu, D. Schonfeld, and R. A. et al. Adaptive Networking for Tele-Immersion. In *Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop, Stuttgart, Germany, 2001*, 2001.
- [9] C. Mirasso, M. Mulder, H. Spoelder, and D. Lenstra. Visualization of the Sisyphus Attractor. *Computers in Physics*, 11(3):282–286, May/June 1997.
- [10] U. Obeysekare, F. Grinstein, and G. Patnaik. The Visual Interactive Desktop Laboratory. *IEEE Computational Science and Engineering*, 4(1):63–71, Jan. 97.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] D. Reed, Giles, and C. Catlett. Distributed Data and Immersive Collaboration. *Communication of the ACM*, 40(11), Nov. 1997.
- [15] 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.
- [16] L. Renambot, H. E. Bal, D. Germans, and H. J. Spoelder. CAVEStudy: an Infrastructure for Computational Steering and Measuring in Virtual Reality Environments. *Cluster Computing journal*, 4(1):79–87, Mar. 2001.
- [17] E. Shaya. XDF, the eXtensible Data Format for Scientific Data. Technical report, ADC/NASA/RITSS, 2000. <http://xml.gsfc.nasa.gov/XDF/XDFwhite.txt>.
- [18] H. J. Spoelder. Virtual Instrumentation and Virtual Environments. *IEEE Instrumentation and Measurement Magazine*, 3(3):14–19, 1998.
- [19] 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.