

CAVEStudy: an Infrastructure for Computational Steering and Measuring in Virtual Reality Environments

Luc RENAMBOT^a Henri E. BAL^{a,b} Desmond GERMANS^b
Hans J.W. SPOELDER^b

^a *Division of Mathematics and Computer Science*

^b *Division of Physics and Astronomy*

Faculty of Sciences, Vrije Universiteit

De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

We present the *CAVEStudy* system that enables scientists to interactively steer a simulation from a virtual reality (VR) environment. No modification to the source code is necessary. *CAVEStudy* allows interactive and immersive analysis of a simulation running on a remote computer. Using a high-level description of the simulation, the system generates the communication layer (based on CAVERNSoft) needed to control the execution and to gather data at runtime. We describe three case-studies implemented with *CAVEStudy*: soccer simulation, diode laser simulation, and molecular dynamics. In addition, we briefly describe a new technique of virtual measuring which closes the loop between simulation, immersive visualization, interaction in simulation domain and steering, giving the scientist deeper insight into the simulated phenomenon.

1. Introduction

High-speed networks and high performance graphics open opportunities for completely new types of applications. As a result, the world of scientific computing is moving away from the batch-oriented management to interactive programs. Also, virtual reality (VR) systems are now commercially available, but so far scientists mainly use them for off-line visualization of data sets produced by a simulation program. This simulation runs on a remote supercomputer without any user-control. 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 [28]; Scientists should be able to measure the data, using a variety of measurement paradigms. This gives rise to a number of research questions: interaction, collaboration, steering, and measurement in virtual reality become central issues in the design of virtual environments for scientific applications.

Networked virtual environments have rapidly developed over the last few years. Numerous toolkits are now available [3,16,24,25]. Each one has some specific features, such as collaboration, portability, and distributed simulations. The most general toolkit, and the one most used, is CAVERNSoft [14]. It provides networking and database functionalities needed in a virtual reality system. Different skeletons of VR application are proposed. The system is open but still requires substantial effort to build new applications.

A related research area is the control of a running simulation, referred to as computational steering [22]. It is defined as interactive control over a simulation during its execution. 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 [4]. However, existing systems, such as SCIRun [22] and CSE [30], are used to build new applications or require modifications to the source code of the application, and thus are unsuitable if the source code is unavailable.

In this paper, we describe a system called *CAVEStudy*, based on the CAVERNSoft toolkit, 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.

The scientist models the simulation as a set of input, output, and graphical objects. 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 a description language, a graphical interface to create description files, 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.

The main contributions of our work are as follows :

- a high-level steering system that does not require modifications to the source code,
- a VR framework to immerse the scientist in the simulation space,
- an experimental evaluation of the steering system and VR framework using various systems, such as an IBM SP2, a 128-node Myrinet cluster computer, and several CAVEs and graphic workstations,
- three real applications (collaborative soccer, laser simulation, molecular dynamics visualization) used as case-studies.

This paper is structured as follows. In Section 2, we survey related research. In Section 3, we introduce our system. Using a small example, we show how it facilitates the coupling of a simulation to a virtual environment. Some applications built with our infrastructure are described in Section 4. Section 4 briefly describes work on virtual measurement. Finally, we present our conclusions.

2. Related work

Our work is related to a wide and active research area referred to as the *Grid*, *Metacomputing*, or the next generation high-performance computing infrastructure [4]. Systems such as *Globus* [5] or *Legion* [8] provide languages, tools, and environments to create new applications that were not conceivable before, such as world-wide collaboration in virtual reality or real-time data-mining of large data sets. Below, we discuss networked virtual environments and steering systems.

2.1. Networked virtual environments

Interactive and collaborative visualization radically change the way scientists use computer systems [4]. With interactive visualization, a user can interact with a program in its visual domain. Distributed collaboration allows multiple

users at different geographic locations to cooperate, by interacting in real-time through a shared application [14,26]. Several toolkits have been designed to provide networked virtual environments. They provide functionalities such as communication, shared-state management, collaboration mechanisms, and 3D graphics.

NPSNET [16] focussed on large-scale virtual environments for battle simulations. The entities simulated are, for instance, tanks, missiles, or soldiers. The database is replicated and split into areas of interest, which permits to reduce the communication. The DIS (Distributed Interactive Simulation, which allows to model remote entities) protocol is used in conjunction with multicast to achieve a better scalability. BrickNet [25] is a toolkit that supports graphical, behavioral, and network modeling of virtual worlds using a shared object paradigm. The network medium is a set of interconnected servers. Objects can be shared by several worlds. The main goal was to build collaborative design environments. MR Toolkit [24] maintains a shared world in a distributed manner, using frequent UDP packets to update the world. Using callback functions, the state updates are propagated across the whole data set. DIVE [3] focus on some network issues to obtain large-scale virtual environments, using mainly reliable multicast and spatial decomposition of the virtual world. Massive [7] is an object-oriented environment that provides multicast, region of interests, 3D graphics support and real-time data streaming. Bamboo [32] focusses on dynamically extensible and real-time networked environments which should configure itself, allowing to add new functionalities or to discover new environments at runtime. Avocado [29], an object-oriented system based on the Performer library, replicates and updates the scene graph among the remote users using a group communication approach. The shared-data structures are kept consistent by some global operation such as deletion or insertion. CAVERNSoft [14] provides networking and database functionalities needed in a virtual reality system. It also comes with different skeletons of VR applications called templates. The system is open but remains very basic. It still requires a substantial effort to build a new application.

2.2. Steering systems

Many existing applications restrict the interaction to the visualization process (e.g., the direction of view, the zoom factor). A more advanced form of in-

teraction, referred to as computational steering, allows the user to interact with the simulation process. Several systems support steering [18], but they typically provide only low-level interactions and require users to monitor or change the application program’s internal variables. We do not address the issue of fine grain steering offered by some systems [11], but we focus on the complete execution of a simulation for a given set of inputs. However, we also examine simulations that produce results during the execution (i.e., iterative methods), shipping the intermediate results as output.

SCIRun [22] is a problem-solving environment for computational science. It provides an integrated framework to construct, steer and study large applications. It has been used in domains such as medicine or geophysics. The scientist constructs the simulation by connecting a set of modules using a data flow paradigm. Each module is a compiled function written in C, C++ or Fortran wrapped in a Tcl/Tk interface. A central module schedules and synchronizes the system. SCIRun allows the scientist to modify the parameters of the simulation, which triggers the evaluation of the connected modules. Some 3D visualization modules are provided to present the data. Such a system is designed to build and test, in a closed loop, new algorithms, but it is not well suited to study existing applications. Moreover, the graphic output is used to visualize results, but not to steer the application. VASE [11] allows the programmer to design a steerable program through the specification of the program structure. This structure is a graph made of blocks of source code and arcs representing the control. The program can be stopped at run-time between blocks and some user scripts (C-like) are executed. At these points, data can be retrieved and modified. The visualization is managed by an external library such as IRIS Explorer. Magellan [31] requires the user to annotate the source code to produce an abstract view of the program, consisting of steering parameters and output data. These objects can be probed, modified at run-time, or provide a breakpoint. A master program can control several steered program. A graphical interface can generate monitoring and steering commands. The visualization has to be done through an external library. CUMULVS [10] is designed to study PVM-based parallel applications. Within the application, data distribution and steering parameters have to be declared. Steering commands are synchronized and applied to all tasks. Moreover, CUMULVS allows several users to connect to the same application. In such a collaboration mode, users can lock some steering parameters. VIPER [23] allows the steering of massively parallel applications, for

instance computational fluid dynamics. VIPER requires the parameters of the simulation to be annotated in the source code. During the execution, the system extracts and transfers data from the application to a remote multi-threaded server. A graphical interface is provided to manage data gathered. Finally, CSE [30] focuses on existing applications. It allows several programs to be integrated within one distributed steering environment. An application is annotated to specify input and output objects, and then connected to a server. This server acts as a data manager for all the components. A component can subscribe to the output of another component. The server manages the data exchange and the synchronization of the whole system. Data management and visualization components are available. Moreover, an editor is available to design graphical data representations. Such a representation is used as direct steering mechanism, instead of a simple classical visualization.

Related to the idea of a virtual workspace for the scientist, the concept of a *virtual laboratory* is explored in research projects like VIDL [20] and Cactus [1]. The authors try to provide a collaborative and problem-solving environment for large-scale simulations (as computational fluid dynamics or relativistic astrophysics). Nevertheless, our main conclusion is that no environment provides all the functionalities needed for an interactive and immersive steering environment, given that we do not want to modify the source code. We designed *CAVEStudy* in that spirit.

3. *CAVEStudy*

Our goal is to build a system that combines the power and the functionalities of computational steering and virtual reality. Such an environment, combining both the control over a simulation and the immersion in the data space, does not exist yet as far as we know.

CAVEStudy mainly consists of two parts: a code generator and a VR framework, as shown in Figure 1. To minimize the programming for the control of the simulation and the data management, the user has to describe the simulation by a description file. This file is processed to generate two objects, a proxy and a server. The simulation is wrapped into a server object to control its execution. The server's interface provides methods to start, stop, pause, and resume the simulation. The data generated by the simulation are automatically propagated

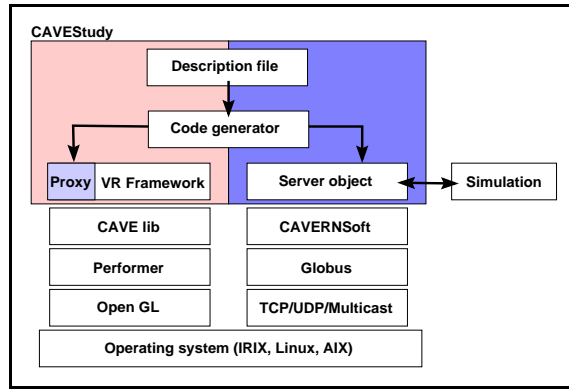


Figure 1. Software layers

to the proxy object. This object can be seen as a local copy of the remote simulation. Through the network, it reflects the input values and the commands to the server. Furthermore, it manages the incoming data from the simulation and presents them to the VR framework.

3.1. Simulation description

To model a simulation, we designed a description language which allows the user to describe the input parameters and the output data of the simulation. It can be seen as a light-weight and dedicated CORBA-like interface description language. Several sections must be present in such a file: a description of the simulation program, a set of input parameters, and a set of output data. Graphical objects can also be described. An example is presented in Figure 2. A graphical interface is available to describe such a file.

To be able to start the simulation, the user must give information about the executable such as its name and directory. This is specified in the **Simulation** section. The way to feed the simulation with the input parameters should also be specified, for example as a command-line or with an input file. The system should know how to acquire the output data produced, for instance on the standard output or from a file. Data manipulated by the simulation can be described using either the provided built-in types or by some types defined by the user. The basic types defined in the system are: *long*, *floating-point*, *string*, *2D-point*, and *3D-point*. New types can be described by combining basic types. An example can be seen in Figure 2 (type *PointTime*). All these types can be used in scalar, list, and matrix objects.

```

Project
{
  Name      BouncingBall
}
Struct PointTime
{
  vector3D  point
  float     time
}
Simulation
{
  Name      bounce
  Executable bounce.exe
  Directory Simulation
  Processes 1
  InputType  cmdline
  InputValue ""
  OutputType file
  OutputValue position.data
}
Input vector3D
{
  Name      init_position
  Value     [ 1.0, 0.0, 0.0 ]
}
Input int
{
  Name      nb_step
  Value     1000
}
Output PointTime
{
  Name      current_position
}
Graphic
{
  Name      the_ball
  Type      sphere
  Value     current_position
  Color     [ 1.0, 0.0, 0.0, 0.0 ]
}

```

Figure 2. A sample description file

Input parameters are described by a name, a type, a dimension (scalar, list, or matrix) and an optional input value when applicable (scalar object). Any number of input parameters can be specified by several **Input** sections. Output objects are given in a similar way using **Output** sections. Finally, it is possible to specify graphical objects. Such an object is the graphical representation of an output data produced by the simulation, and its type can be selected among several ones (sphere, line, surface, etc). The user is able to interactively modify the values of input parameters to steer visually the simulation.

3.2. Code generation

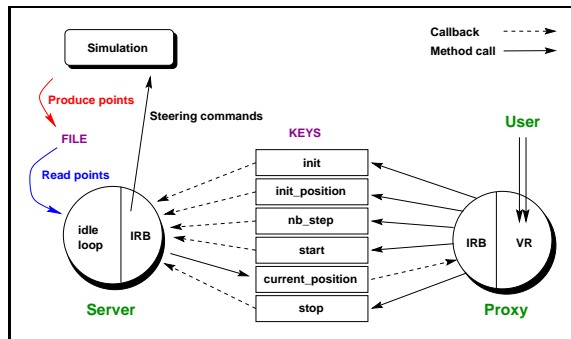


Figure 3. A resulting architecture

We wrote a code generator for the description files. It generates C++ code for the CAVERNSoft [14] network layer, as shown on the right part of Figure 1. We selected CAVERNSoft because it provides functionalities to build networked virtual environments and because it is widely used in the VR community. CAVERNSoft, based on the Nexus communication library from Globus, uses a “publish and subscribe” paradigm. A site can define keys to publish its own data, and a remote site that subscribes to these keys will automatically receive the data through callback functions. This mechanism can be used for small data (tracker data) to large data sets (data-mining) using different policies. CAVERNSoft supports persistence, thread management, and network protocols (TCP/UDP/multicast). It also provides some facilities for avatars and 3D models using the SGI *Performer* library. Currently, our code generator can produce programs for CAVERNSoft version 1 and version 2 (also known as G2). The source code generated can be used on SGI Irix, Linux, IBM Aix, and Microsoft Windows.

Our code generator produces C++ classes for the server object and the proxy object. Each of these objects contains a threaded IRB (network object of CAVERNSoft) and defines a set of keys with their associated callback functions. For each input or output object, a key is defined to transmit the value. The marshaling code for all the types is generated to be able to use our system in a heterogeneous environment. A set of keys is also created for the control of the simulation (*initialize*, *start*, *stop*, *pause*, *resume*, *shutdown* methods). It is therefore possible to manipulate proxy and server entities as C++ objects,

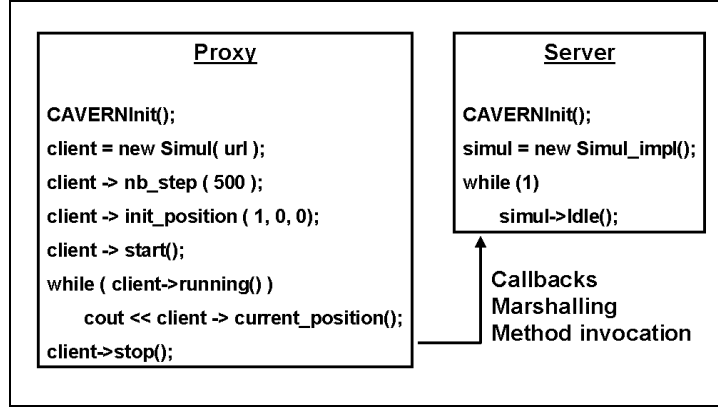


Figure 4. Server and proxy objects

without dealing with network issues. Figure 3 presents the resulting architecture corresponding to the description file of Figure 2. It shows the relation between the server object (see in Figure 1) and the proxy object. Figure 4 shows an example of the resulting programs generated by *CAVEStudy*. We present a clean and simple interface of the two components to the programmer. All the communication aspects are hidden within the implementation of C++ objects. The only public methods are those concerning the setting of new values for the input parameters and those to get the values from the simulation. For the server program (cf. Figure 4, on the right), we create an instance of the server class and enter an endless loop waiting for remote method invocations. The proxy (cf. Figure 4, on the left) connects to a given server (machine name plus port number) and sets the values of the input parameters. It is then possible to control the execution of the remote simulation with some method invocations (*start*, *stop*, etc). The data produced by the remote simulation are sent and stored automatically to the proxy object, and can be read at any time by local method calls (without any communication). This proxy object is embedded into our VR framework.

	TPC	<i>CAVEStudy</i>	TCP	<i>CAVEStudy</i>	Throughput
	Latency	Latency	Throughput	(char)	(double)
Ethernet 100 Mbits	0.20 msec	1.64 msec	7.9 Mbytes	5.1 Mbytes	2.7 Mbytes
Internet	3.71 msec	7.78 msec	1.7 Mbytes	1.5 Mbytes	1.3 Mbytes

Figure 5. Performance measurement

We present in Figure 5 some preliminary performance results produced by the current (non-optimized) prototype. The round-trip latency measurement is performed using the TCP protocol with a one-byte buffer in a RPC manner (triggering a function which provides the data received). The throughput test is done with an 8 Kbytes buffer using either character data or floating-point values (requiring marshaling and network-independent representation). Table 5 shows the latency and the throughput between two nodes of a Linux cluster using 100Mbits Ethernet, and the same experiments between two computers within Amsterdam using Internet. The callback mechanism and the data marshaling of CAVERNSoft introduce some performance loss, mainly on the local network. Using Internet, the *CAVEStudy* latency is approximately twice the one using a low-level programming approach. The differences are smaller for the throughput: it decreases from 7.9 Mbytes per second using a low-level TCP socket to 5.1 Mbytes per second with *CAVEStudy* using Ethernet. Applying a packing algorithm to deal with heterogeneous systems (big-endian and little-endian representation) gives us a throughput of 2.7 Mbytes per second using Ethernet. Between two sites interconnected by Internet, the *CAVEStudy* overhead remains minimal. Even though our main focus in this work is not the performance, we consider these results encouraging for a first implementation.

3.3. VR framework

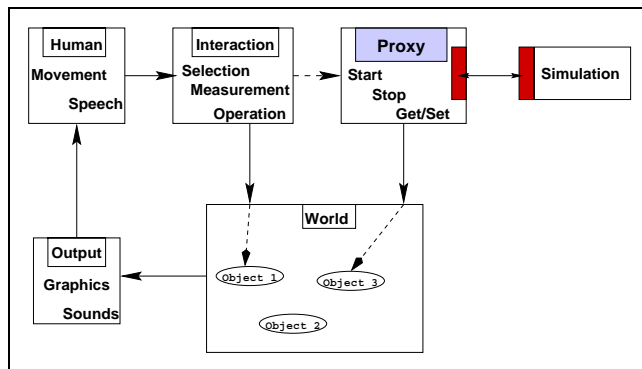


Figure 6. VR framework

Besides the code generator, we developed a virtual reality framework to steer and visualize the data produced by the simulation. It is built on top of

the *Performer* library to exploit the high performance of SGI workstations and on top of the *CAVELib* library to control VR devices (multi-screens and tracker management), as shown on the left part of Figure 1. The architecture is described in Figure 6, focus-sing on the VR framework as referred in Figure 1. It consists of a shared-world where the objects of the simulation are represented. These objects are updated through the generated proxy object. An interaction module allows the user to send commands to the proxy or to directly manipulate the objects of the simulation to steer it. This framework is functional but still under development, following the needs of the applications described in the next section. For instance, we developed a menu system to control the execution of the simulation. Using sliders and buttons it is possible to modify parameters values. The interaction modules allows the user to scale, rotate and translate objects. The graphic part includes basic objects (lines, point, spheres) using color, transparency and textures. A board provides also textual informations. More specific details are discussed in the application section. Figure 7 shows the default virtual environment provided by *CAVEStudy*. It consists of a virtual laboratory with a manipulation box where data can be inserted, and a board to present some text information.

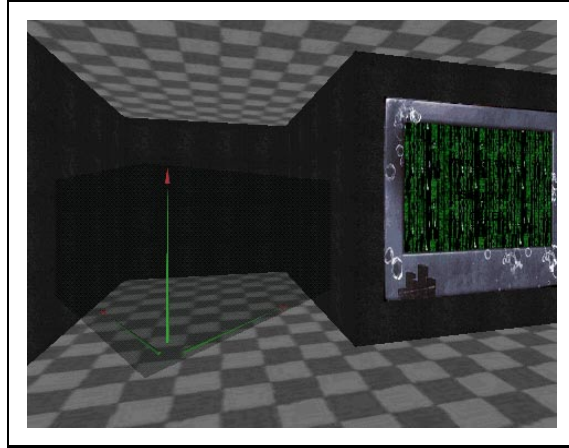


Figure 7. *CAVEStudy* laboratory

The current version of our VR framework was tested and used with two distinct CAVEs, an Immersadesk, SGI workstations, and finally Linux PCs (using *CAVELib* and *Performer* ports on Linux). A new version is under development to support Microsoft Windows, using either *DirectX* or *OpenGL*.

4. Applications

To evaluate our approach of coupling a simulation and a virtual-reality environment, we implemented three different applications using *CAVEStudy: Interactive Soccer*, *Diode Laser Simulation*, and *Molecular Dynamics*. We use these to illustrate the ease of incorporating an existing application, the usability of such a method, and the added value for the user.

4.1. Interactive RoboCup

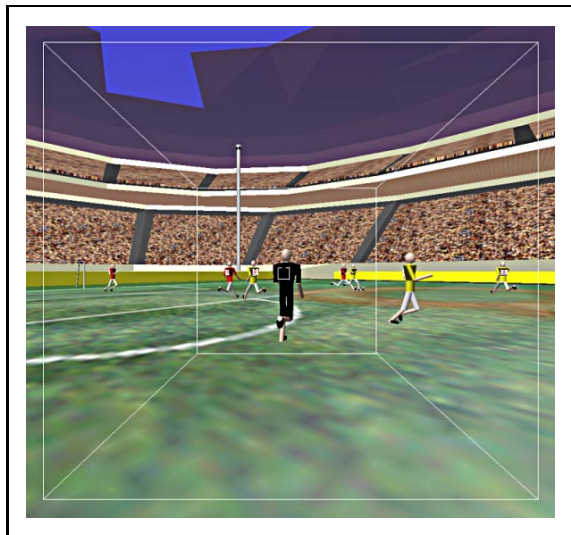


Figure 8. Interactive RoboCup

RoboCup (Robot Soccer) is a standard problem from Artificial Intelligence [12]. Its goal is to let teams of cooperating autonomous agents play a soccer match, using either real robots or simulated players. We constructed a VR environment in which humans in CAVEs at different geographic locations can play along with a running RoboCup simulation in a natural way. A central role is played by the so called *Soccer Server*, which keeps track of the state of the game and provides the players with information on the game. The players are individual processes that can request state information from the server and autonomously compute their behavior. The server also enforces the rules of RoboCup and ignores invalid commands from the players.

Our RoboCup VR system uses the unmodified existing server software. The players communicate with the server by sending soccer commands. The commands are expressed in a simple language, consisting of accelerations, turns, and kicks. The server discretizes time into slots and only the last command of a player within a time slot is executed. Also, the kick command requires the player to be close to the ball. We run such a system on a Myrinet cluster computer.

The CAVE program allows the user to be immersed in the game and to interact with it. We implemented a proxy which uses the same information and communication as the existing 2D visualization. The data set described in our *CAVEStudy* configuration file consists mainly of the player positions and the ball position. From successive states of the game, the visualization system computes several quantities such as direction, velocity and acceleration of the players. We built a virtual stadium (Figure 8) and a parameterized soccer player whose movements are interpolated between three different modes: standing still, walking, and running. We developed software to track the behavior of a human in the CAVE. One tracker is connected to the viewing glasses and monitors positional changes of the human player inside the CAVE. The second tracker is connected to the wand (a 3D mouse), which is used for global movements over the soccer field. The third tracker is attached to the foot of the human player and is used to recognize a kick. We convert tracker changes into soccer commands and transmit these to the server. Finally, we coupled two CAVEs located in Amsterdam and Stockholm [27]. Each CAVE is connected to the same proxy, so the two humans participate in the same game.

The most difficult problem in realizing a virtual RoboCup system is caused by the latency of the simulation program. If the human player moves over the virtual soccer field, these moves happen almost instantaneously for the human. In contrast, the soccer server will require some time to process the change of position. Also, the wide-area (Internet) connection causes a substantial delay. This problem is a typical example of how a delay introduced by a simulation program can harm a natural and real-time interaction [21]. We are currently developing accurate and low-bandwidth algorithms for the navigation (walking across the soccer field) and the interaction (kicking detection) that generate commands to the server.

The first implementation of this system was done without *CAVEStudy*. We had to program the communication between the different components, which is tedious and error prone. Using *CAVEStudy*, the communication is automatically

generated. More generally, the case study with RoboCup shows the applicability of *CAVEStudy* on the large class of agent/server systems.

4.2. Diode laser simulation

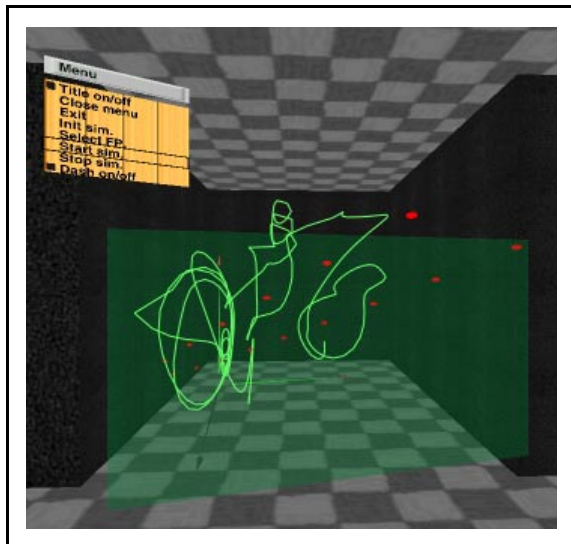


Figure 9. Diode laser simulation

Simple systems with only a few degrees of freedom can exhibit chaotic dynamical behavior, even when the evolution is deterministic. A precise knowledge of the initial conditions allows us, in principle, to calculate exactly the future behavior of the system. One problem in understanding a chaotic system is the reconciliation of the apparent conflict between randomness and determinism. An application of this class we implemented is the visualization of a diode laser behavior, referred to as the *Sisyphus Attractor* [17]. 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. 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 9.

A previous study [17] 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.

4.3. Interactive Molecular Dynamics

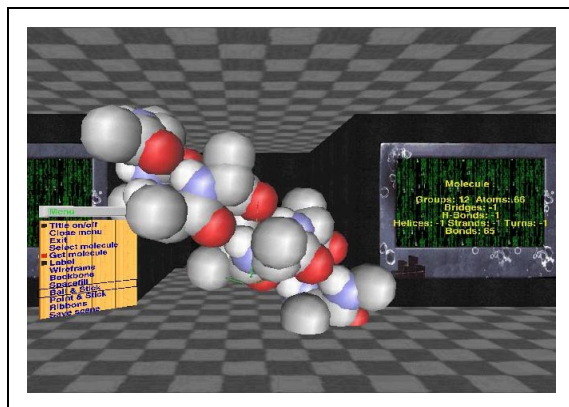


Figure 10. Interactive Molecular Dynamics

Our third application concerns the coupling of a molecular dynamics (MD)

simulation to a virtual reality system. Molecular modeling tools are essential to design and study new molecules. For example, when steering a molecular dynamics simulation, the user can express external forces to help the system to overcome energy barriers, or can help in the search for likely geometric configurations in docking problems. Many studies focus on numerical simulations or visualization tools, and some of them describe the advantages of connecting molecular dynamics simulation to visualization. For instance, NAMD [19], which includes state-of-the-art serial and parallel algorithms, has been designed as a flexible program, incorporating many options such as control integration methods, force field parameters, and restart capabilities. VMD [9] is a visualization environment for structural biology which has been widely used for plain visualization, docking studies, structure refinement or trajectory analysis. An important feature is that it is possible to couple a NAMD running simulation to VMD to study trajectories of molecules. Moreover, some local forces can be applied using either a 2D graphical interface or a 3D haptic feedback tool [9]. In [13], the authors describe how they link their molecular simulation to VMD. The user can express external forces to help the system to overcome energy barriers between states. Another example is the molecular docking simulation program described in [15]. A user in a virtual reality environment can interact with a genetic algorithm running on a parallel computer to help in the search for likely geometric configurations.

VR allows the scientist to gain a deeper understanding of the complex conformations in 3D. Moreover, modifying 3D structures or expressing forces is intrinsically a 3D process, for which the use of an immersive virtual environment is a perfect match. As a feasibility study (and for later experiments on interaction and measurement), we wrapped the molecular dynamics NAMD [19] simulation and visualized it in a VR environment.

Our current implementation allows a remote simulation running on the DAS parallel cluster computer [2] to be visualized in the CAVE. The input parameters we selected are the name of the molecule on which the simulation will be applied, the number of time steps of the simulation, and the temperature. It corresponds to the minimal set of parameters among the large possibilities of NAMD. We did not implement the interactive parameter selection yet, but parameters can be modified at starting time. As output of the simulation, we use the PDB description files produced as intermediate result during the execution. These files, which contain the position and velocity of all the atoms, are read by the proxy process and sent continuously to the visualization, showing the dynamics

of the molecule. Several classic molecule representations are available (wireframe, sphere, backbone). An example is shown in Figure 10.

The coupling of MD simulation to visualization has already been done before, but always by modifying source code. Usually, they are made by the original developers modifying their own software and using simple TCP/IP connections, and in a very application-specific manner. Using *CAVEStudy*, we were able to very quickly couple NAMD to our virtual environment. With *CAVEStudy*, we can easily switch between several simulation packages. Furthermore, to steer such a simulation adequately, 3D forces should be expressed, which can efficiently be done in a 3D VR environment implemented by *CAVEStudy*.

5. Measuring

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 (modification of the source code), or are batch-oriented by nature (extensive search).

We are working on a system based on *CAVEStudy* for measuring and steering using virtual reality environments [6]. It will allow non-expert users to use virtual reality environments as a measuring tool next to a visualization tool. A few examples are : measuring the pressure directly onto the visualization during a gas simulation, measuring fields or electric forces within a molecular dynamics simulation, or measuring distances in pre-surgery medical applications. This system will profit from all the benefits of *CAVEStudy* (no modification of the simulation program and interaction with the simulation's original input and output). The measuring process will be done interactively by the final user, within the VR world. The measuring methods will be provided by our VR environment, operating on the data produced by the steered simulation. Such a system raises numerous research issues, among them virtual measurement paradigms, comparison with other steering methods, calibration and validity of the measurements.

We are currently studying two relatively simple simulations, a charge-recombination simulation, and an ideal-gas simulation. For both case-studies we try to show that it is possible to measure physical quantities and derived

quantities from data in the visual domain.

The coupling of a steering system to some measuring techniques will produce a high-level and easy-to-use steering environment, putting the scientist within an “instrumentation loop”.

6. Conclusion and Future work

In this paper, we described the *CAVEStudy* system that allows the scientist to interactively steer a simulation, without requiring any modification to the original program. It enables an interactive and immersive analysis of a simulation running on a remote computer.

A set of input, output, and graphical objects are specified by the scientist in a description file. These objects represent the input parameters of the simulation and the data produced. Using such a description, our system generates a server to control the simulation and to send data produced to the virtual environment. These data are used to update the graphical objects, which can be manipulated to steer the simulation. Thus, it is possible to visualize and control the program directly in the domain space of the simulation. We run simulations on various systems, such as an IBM SP2 and a Myrinet cluster computer, and coupled them to several CAVE VR systems. Moreover, three real applications (soccer, laser simulation, molecular dynamics visualization) have been studied. The lessons learned from these applications are that different types of applications can easily be implemented using *CAVEStudy*, that *CAVEStudy* is easy to use and to maintain compared to previous methods, and that 3D interactive visualization and steering help to gain a better insight in a complex system.

In our future work, we will develop *CAVEStudy* towards several directions. First, we plan to incorporate a better data management allowing the user to deal with data fluxes coming from the simulation. With the applications already implemented, we observed several patterns: data generated once, small data produced continuously, large data sets sent several times during the simulation, several data fluxes produced synchronously. It would be useful to the scientist to visualize these data sets independently of the simulation time. We can imagine that the user will be able to “walk” through the simulation time in a VCR manner (forward, backward, rewind, etc), while the remote program generates the data at its own speed. We will also design and implement the

measuring techniques sketched in the last section, to achieve a higher level of steering with *CAVEStudy*. This environment will provide a generic set of interaction, selection, and measuring functions. We will apply these on some new applications like medical simulations and robot exploration.

The whole *CAVEStudy* system will be available on our web site:

<http://www.cs.vu.nl/~renambot/vr>

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*, Pittsburgh, PA, August 2000. IEEE Computer Society Press.
- [2] H. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, T. Rühl, and F. Kaashoek. Performance Evaluation of the Orca Shared Object System. *ACM Transactions on Computer Systems*, 16(1):1–40, February 1998.
- [3] Christer Carlsson and Olof Hagsand. DIVE — A Platform for Multi-User Virtual Environments. *Computers and Graphics*, 17(6):663–669, November–December 1993.
- [4] A. Foster and C. Kesselman. *The Grid: Blueprint for a New Computer Infrastructure*. Morgan Kaufman, 1998.
- [5] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.
- [6] Desmond Germans, Hans J.W. Spoelder, Luc Renambot, and Henri E. Bal. High-Level Steering: Measuring in Virtual Reality Environments. Technical report, Vrije Universiteit, Faculty of Sciences, september 2000.
- [7] Chris Greenhalgh and Steven Benford. MASSIVE: A collaborative virtual environment for teleconferencing. *ACM Transactions on Computer-Human Interaction*, 2(3):239–261, 1995.
- [8] Andrew S. Grimshaw and William A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.
- [9] William F. Humphrey, Andrew Dalke, and Klaus Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.
- [10] G.A. Geist II, J.A. Kohl, and P.M. Papadopoulos. CUMULVS: Providing fault tolerance, Visualization, and Steering of Parallel Applications. *Int. J. of Supercomputer Applications and High Performance Computing*, 3(11):224–235, 1997.
- [11] D.J. Jablonowski, J.D. Bruner, B. Bliss, and R.B. Haber. VASE: The Visualization and Application Steering Environment. In *Proceeding of Supercomputing'93*, pages 560–569, 1993.
- [12] Hiroaki Kitano, Manuela Veloso, Peter Stone, Milind Tambe, Silvia Coradeschi, Eiichi Osawa, Itsuki Noda, Hitoshi Matsubara, and Minoru Asada. The RoboCup Synthetic A-

- gents Challenge 97. In M. Pollack, editor, *15th International Joint Conference on Artificial Intelligence*, pages 24–29, 1997.
- [13] Jonathan Leech, Jan F. Prins, and Jan Hermans. SMD: Visual Steering of Molecular Dynamics for Protein Design. *IEEE Computational Science & Engineering*, 3(4):38–45, Winter 1996.
- [14] J. Leigh, A.E. Johnson, T.A. DeFanti, and M. Brown. A Review of Tele-Immersive Applications in the CAVE Research Network. In *IEEE Virtual Reality'99*, pages 180–187, 1999.
- [15] D. Levine, M. Facello, P. Hallstrom, G. Reeder, B. Walenz, and F. Stevens. Stalk: An Interactive System for Virtual Molecular Docking. *IEEE Computational Science*, 4(2):55–65, April-June 1997.
- [16] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A Network Software Architecture for Large-Scale Virtual Environment. *Presence*, 3(4):265–287, 1994.
- [17] C.R. Mirasso, M. Mulder, H.J.W. Spoelder, and D. Lenstra. Visualization of the Sisyphus Attractor. *Computers in Physics*, 11(3):282–286, May/June 1997.
- [18] J.D. Mulder, J.J. van Wijk, and R. van Liere. A Survey of Computational Steering Environments. *Future Generation Computer Systems*, 13(6), 1998.
- [19] Mark T. Nelson, William F. Humphrey, Attila Gursoy, Andrew Dalke, Laxmikant V. Kalé, Robert D. Skeel, and Klaus Schulten. NAMD: A Parallel Object-Oriented Molecular Dynamics Program. *The International Journal of Supercomputer Applications and High Performance Computing*, 10(4):251–268, 1996.
- [20] U. Obeyesekere, F.F. Grinstein, and G. Patnaik. The Visual Interactive Desktop Laboratory. *IEEE Computational Science and Engineering*, 4(1):63–71, Jan. 97.
- [21] K. Shin Park and R.V. Kenyon. Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment. In *IEEE Virtual Reality'99*, pages 104–111, 1999.
- [22] S.G. Parker, M. Miller, C.D. Hansen, and C.R. Johnson. An Integrated Problem Solving Environment: the SCIRun Computational Steering System. In *Hawaii International Conference of System Sciences*, pages 147–156, Jan. 1998.
- [23] S. Rathmayer and M. Lenke. A Tool for On-line Visualization and Interactive Steering of Parallel HPC Applications. In *Proceedings of the 11th IPPS '97*, pages 181–186, 1997.
- [24] Chris Shaw, Mark Green, Jiandong Liang, and Yunqi Sun. Decoupled Simulation in Virtual Reality with the MR Toolkit. *ACM Transactions on Information Systems*, 11(3):287–317, July 1993.
- [25] G. Singh, L. Serra, W. Png, and Hern Ng. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence*, 3(1):19–34, 1994.
- [26] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.
- [27] Hans J. W. Spoelder, Luc Renambot, Desmond Germans, Henri E. Bal, and Frans C.A. Groen. Man Multi-Agent Interaction in VR: a Case Study with RoboCup. In *Poster Session - IEEE Virtual Reality'00*, 2000.

- [28] Hans J.W. Spoelder. Virtual Instrumentation and Virtual Environments. *IEEE Instrumentation and Measurement Magazine*, 3(3):14–19, 1998.
- [29] Henrik Tramberend. Avocado : A Distributed Virtual Reality Framework. In *IEEE Virtual Reality'99*, pages 14–21, 1999.
- [30] J.J. van Wijk and R. van Liere. *An Environment for Computational Steering*. Computer Society Press, 1997.
- [31] J. Vetter and K. Schwan. High Performance Computational Steering of Physical Simulations. In *Proceedings of the 11th IPPS '97*, pages 128–132, 1997.
- [32] K. Watsen and M. Zyda. Bamboo – A Portable System for dynamically Extensible, Real-Time, Networked, Virtual Environments. In *IEEE Virtual Reality Annual International Symposium (VRAIS'98)*, 1998.