

# High-Level Steering : Measuring in Virtual Reality Environments

Desmond GERMANS<sup>1</sup>

Hans J.W. SPOELDER<sup>1</sup>

Luc RENAMBOT<sup>2</sup>

Henri E. BAL<sup>2,1</sup>

<sup>1</sup>Division of Physics and Astronomy

<sup>2</sup>Division of Mathematics and Computer Science

Faculty of Sciences, Vrije Universiteit

De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

## Abstract

*We present a new paradigm that combines computational steering and measuring techniques in virtual environments. It enables the scientist to interactively steer a running simulation at high level. Our steering environment does not require modification to the simulation's source code, using a description of the simulation. Easy to use, the scientist can get a deeper insight into the simulated phenomenon. The measuring paradigm closes the loop between simulation, immersive visualization, interaction in simulation domain and steering. We describe two case-studies implemented with CAVEStudy where measuring techniques are incorporated: an ideal-gas simulation and a charge-recombination simulation. On these examples, we validate the measuring techniques showing it is possible to measure from the visual domain.*

## 1. Introduction

In recent years, the availability of virtual reality and high-performance computing has opened opportunities for completely new types of applications. Traditionally, scientific visualization displays computations (performed on a remote supercomputer) in an off-line and non-interactive fashion. With high-speed networks and high performance graphics, the world of scientific visualization is moving away from off-line and batch-oriented management to a tighter visual style of processing, using new interactive virtual reality methods and computational steering [5, 16, 18]. 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 [24]; Scientists should be able to *measure* the

data, using a variety of measurement paradigms. This gives rise to a number of new research questions.

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.

In this paper, we sketch a system for measuring and steering using virtual reality environments. The system supports interactive and immersive analysis and control of a simulation running on a remote computer. It allows non-expert users to use virtual reality environments as a measuring tool next to a visualization tool. Furthermore, the system does not alter the simulation program (as this is not always possible) but interacts with the simulation's original input and output [19]. With the system, we address virtual measurement paradigms, comparison with other steering methods and special issues that arise, like calibration and tolerance of measuring in VR.

To test the system, we study two relatively simple simulations, a charge-recombination simulation, and an ideal-gas simulation. For both case-studies we show that it is possible to measure physical quantities and derived quantities from data in the visual domain.

The main contributions of our work are as follows :

- We present a new paradigm that enables scientists to measure quantities and derived quantities in the visual domain in virtual reality environments.
- We validate the concept of measuring in virtual environments with two existing simulations: a ideal-gas simulation and a charge recombination simulation.
- We show the usefulness of the coupling of a steering system to some measuring techniques to obtain a high-level and easy-to-use steering environment.

This paper is structured as follows. In Section 2, we survey related research. In Section 3, we introduce our steering system, referred to as *CAVEStudy*. Our measuring paradigm is described in Section 4, and applied on two existing simulations in Section 5. 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 [5]. Systems such as *Globus* [6] 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.

Interactive and collaborative visualization radically change the way scientists use computer systems [5]. 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 [23]. Several toolkits have been designed to provide networked virtual environments [3, 7, 12, 13, 21, 22, 25, 28]. They provide functionalities such as communication, shared-state management, collaboration mechanisms, and 3D graphics. The idea of a *virtual laboratory* is explored in some research like VIDL [14] and Cactus [1], trying to provide a collaborative and problem-solving environment for large-scale simulations (as computational fluid dynamics or relativistic astrophysics).

Many existing applications restrict the interaction to the visualization process (e.g., the direction of view, the zoom factor). A more advanced form of interaction, referred to as computational steering, allows the user to interact with the simulation process. Several systems support steering [10, 11, 15, 17, 26, 27]. But they typically provide only low-level interactions and require users to monitor or change the application program’s internal variables. 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.

Our main conclusion is that no environment provides the functionalities needed for an immersive steering and measuring environment, given that we do not want to modify the source code. Adding measuring functionalities to our VR steering environment *CAVEStudy* [19] should let the user to explore efficiently his simulation and to get a deeper insight of the studied phenomena.

## 3. Steering

We built a system, *CAVEStudy* [19], that combines the power and the functionalities of computational steering and virtual reality. Our environment combines both the control over a simulation and the immersion in the data space, as depicted in Figure 1. The simulation is seen as a *black box* that takes input parameters and produces data. These data sets are presented to the user in a virtual world where it is possible to interact and alter them. The modifications are translated, if needed, seamlessly to new parameter values for the simulation.

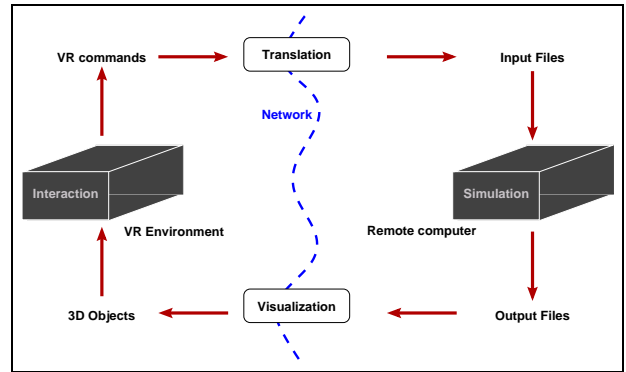


Figure 1. Steering in a VR environment

*CAVEStudy* mainly consists of two parts: a code generator and a VR framework, as shown in Figure 2. To minimize the programming for the control of the simulation and the data management, the user has to describe the simulation with 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 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. 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. The user must provide 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 param-

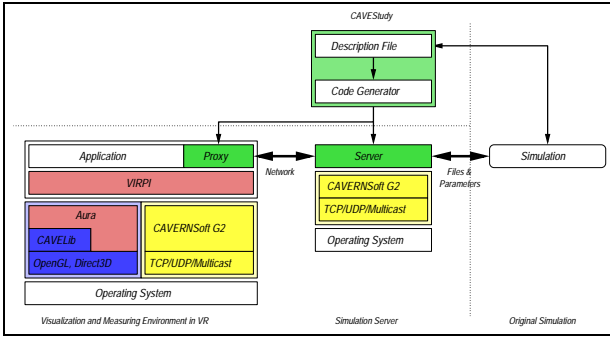


Figure 2. Overall architecture

eters 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 are: *long*, *floating-point*, *string*, *2D-point*, and *3D-point*. New types can be described by combining basic types. All types can be used in scalar, list, and matrix objects. Input parameters are described by a name, a type, a dimension (scalar, list, or matrix) and an optional initial value. 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. The user is able to interactively modify these objects, steering visually the simulation. The value of such an object should be an output object, and the type can be selected among several ones (sphere, line, surface, etc).

### 3.2. Code generation

We wrote a code generator for the description files. It generates C++ code for the CAVERNSoft [12] network layer, as shown on the right part of Figure 2. 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. It also supports persistence, audio and avatar management,

Our code generator produces objects for the server and the proxy. Each of these objects contains a set of keys with

their associated callback functions to transmit input or output values. 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, without dealing with network issues. For the server, we generate an program which is an endless loop waiting for remote method invocations. The proxy object is embedded into our VR framework.

### 3.3. VR framework

Besides the code generator, we developed a virtual reality framework to steer and visualize the data produced by the simulation, as shown on the left part of Figure 2. It is built on top of the *Aura* library which alleviates portability issues among several operating systems (Win32 and different flavors of Unix). The *Aura* library is designed to efficiently exploit standard 3D libraries (*OpenGL* and *DirectX*) across a large variety of machines.

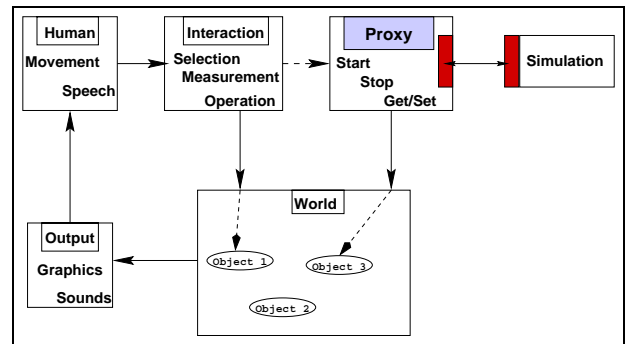
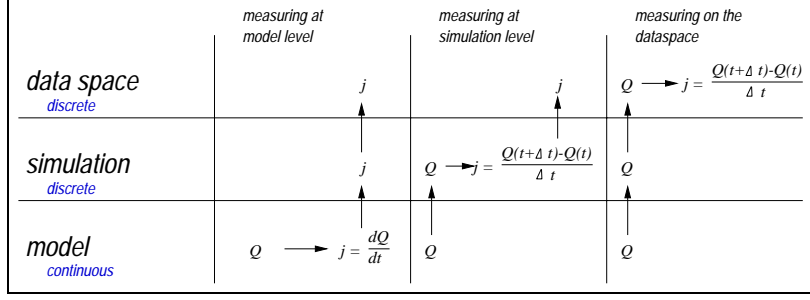


Figure 3. VR framework

VR aspects (multi-displays and tracker management) are managed by the virtual reality-part of *Aura*. This highly reconfigurable part currently supports two modes : first, a “simulated” mode providing a simulator for a virtual reality environment on a standard workstation, and second, a “real” mode using the *CAVELib* library to control a real VR device. On top of *Aura* resides *VIRPI*, an extendable GUI-library, providing interaction tools, widgets and windows for virtual reality environments. The architecture is described in Figure 3, focussing on the VR framework as referred in Figure 2. 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



**Figure 4. Example of measuring for an electrical charge distribution simulation**

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. The default environment provided by *CAVEStudy* consists of a virtual laboratory with a manipulation box where data can be inserted, and a board to present some text information.

#### 4. Measuring

Measuring can be defined as a quantification of an observation in a given space. We see a virtual environment which displays the simulation results as space in which we can observe, and on which we can perform measurements. Data spaces can either be (multi)scalar or vector. In this paper, we will only consider the former type. Thus, the dataspace with which we work is the three dimensional subset of a possibly higher dimensional scalar space. The space can be either static (single set of results) or dynamic (running simulation). Measuring of scalar data can be split in the following categories:

- counting;  $N = \#\{P(x, y, z) | P(x, y, z) > P_{thres}\}$
- summing;  $S = \sum_V P(x, y, z)_i$
- averaging;  $\bar{P} = \frac{S}{N}$
- derivatives with respect to space or time

For all of these quantifications, a selection of the total space is needed. Counting is done within a volume or through a surface, as is summing and averaging.

As an example, consider a scalar field consisting of electrostatic charges. A current is the time-derivative of the charge distribution:

$$\vec{j}(\vec{x}, t) = \frac{\partial Q(\vec{x}, t)}{\partial t}$$

It can viewed at three levels (Figure 4). The first level is the analytical or model-level. Here, measuring the current means directly deriving an expression for it from theory. Then, the simulation calculates the current and from the visualization we can immediately read the result. The second level is the discrete level. Measuring at this level means that the simulation derives the current from the calculated discrete charge distribution, and passes it up to the visualization, which can again immediately display the result. Note that here we introduce an error by approximating the continuous analytical derivative by a discrete differential quotient:

$$\vec{j}(\vec{x}, t) = \frac{Q(\vec{x}, t + \Delta t) - Q(\vec{x}, t)}{\Delta t}$$

The quality of this approximation is related to the length of the time step  $\Delta t$  that is used in the simulation. Measuring at the last level means that calculation of the current is done from the data space. Again we approximate the analytical derivative by a differential quotient.

Measuring at the first level implies mathematically circumventing specific problems. Practically, this would mean restructuring and rewriting the simulation program each time the scientist wants to measure something new. Measuring at the second level means augmenting the simulation program with probing instructions, and still requires the scientist to have knowledge of the way the simulation is programmed. Measuring at the third level, however, leaves the simulation untouched and puts the measurement calculations on the visualization and interaction side, close to the scientist.

Measuring at the visualization side has several other advantages. When the scientist wants to change the setup of the measuring tool (for instance, changing the volume in which particles are being counted, changing the plane on which a force is calculated, etc.), response is more flexible and faster if this only involves changing visualization-side parameters. This feature is particularly useful for rapid prototyping. Furthermore, measuring in the temporal domain

automatically suffers less from latency effects when measurement is done locally.

#### 4.1 Interaction

Measuring in virtual environments requires accurate and intuitive interaction with both the visualized data and the measuring tools. Scientists need to be able to express accurately what subset of the data they want to measure (points in a volume, flux through a plane, etc.), and, what operation they want to apply on the subset (count, integrate, distance, etc.). The virtual environment should supply both intuitive tools to measure the data, as well as a simple and clear programming environment capable of expressing other means of interaction. In this paper, we consider selection by means of a volume. The user can position and scale the volume.

#### 4.2 Results, Calibration and Validity

Measurement results can be visualized either by overlaying them onto the existing data, or by generating new datasets with new visualizations. An example of the first case is a particle system where streams and vortices are being investigated. The measurement results in vector-arrow glyphs pointing in the direction of movement of the particles, projected in the same space. When measurement results are gathered in a new dataset, this dataset can again be used for measuring.

To validate a virtual measuring tool, we essentially want to compare measuring from the data space with the mathematical model behind the simulation (indicated by the top and bottom bar in Figure 4). We can say that measuring from the data space is accurate if the results lie within statistical tolerances around the model predictions. In the charge distribution example, the accuracy is related to the size of the time step, and the size of the volume used to select the charges (in case a subset is being probed).

#### 4.3 The Instrumentation Loop

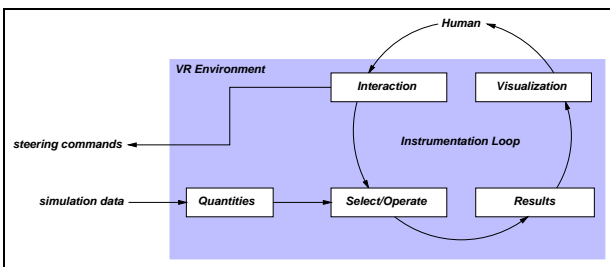


Figure 5. The Instrumentation loop

Our system sets up what can be called *the instrumentation loop*. Figure 5 shows an outline of the system. The

user (*Human*) sees the visualized data in the virtual reality environment (*Visualization*). Using this, the human specifies subregions (for instance, positions a probe volume) and sets the simulation parameters (*Interaction*). These parameters are sent to the simulation (so it can be required, or even restarted with different parameters). The probe selects a subset of the data, and from this the wanted measurements are calculated (*Select/Operate*). These results are again visualized so the user can re-iterate until he is satisfied with the measurements. As explained above, this instrumentation loop is tight and runs local on the machine(s) providing the virtual environment. This way, interaction can be immediate, even though the simulation might take a long time (on one or more remote machines) to process.

### 5. Case Studies

To evaluate our approach, we performed two case studies, a ideal-gas simulation and a charge-recombination simulation [20]. Both experiments run remotely on a node of a parallel cluster [2] and communicate via network with a CAVE environment [4] using CAVEStudy [19]. The data for the particles (for both simulations) consists of a list of particle coordinates for each discrete time step. From this list, the particles are visualized using a set of points. Furthermore, for the charge recombination simulation the color of the points indicate the charge. The case studies illustrate the ability to measure directly from the data space in a virtual reality environment.

#### 5.1. Study 1: Gas Simulation

An ideal gas is a set of moving particles confined in a volume  $V$  that only experience the boundaries of the volume (they reflect on its surface); there is no interaction between the particles themselves. The ideal gas adheres to basic thermodynamic macroscopic laws (like  $pV = RT$ ). In our setup, the user can manipulate either a cubic or spherical volume (indicated by a transparent cube or sphere in Figures 7(a) and 7(b)) through the gas. For this volume, the user can choose the size arbitrarily. The measuring tools then count the number of particles, as well as calculate the particle pressure within the volume.

The pressure of an ideal gas can be obtained by measuring the particle flux along the surface of the probing volume. The microscopic pressure is defined as the impulse transfer of the particles on the surface of a volume:

$$\begin{aligned}
 p &= 2 \sum_V \frac{F_n}{A} \\
 &= 2 \sum_V \frac{ma_n}{A}
 \end{aligned}$$

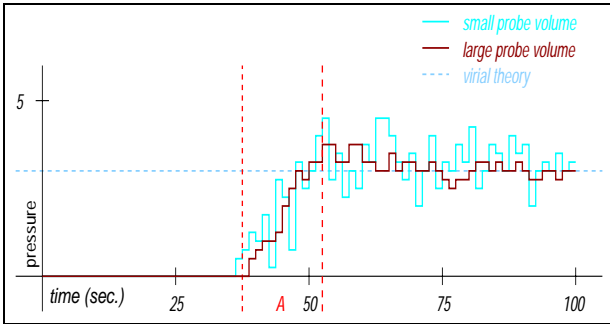
$$\begin{aligned}
&= 2 \sum_V \frac{m}{A} \frac{dv_n}{dt} \\
&\approx 2 \sum_V \frac{m}{A} \frac{v(t - \Delta t) - v(t)}{\Delta t}
\end{aligned}$$

where  $p$  is the pressure of the gas,  $A$  is the surface of the volume  $V$ ,  $F_n$  is the force applied by particle  $n$  perpendicular to the surface,  $m$  is the mass of the particle (we assume all particles to be of identical mass) and  $v_n$  is the speed of particle  $n$ , again perpendicular to the surface.

Also, for the entire system, we can describe the pressure using the virial rule [9]. The virial rule states that the pressure of a system in equilibrium is:

$$p = \frac{2}{3V} [E_{kin} - \Xi]$$

where  $V$  is the total volume,  $E_{kin}$  is the trajectory-averaged kinetic energy of all particles, and  $\Xi$  is the trajectory-averaged virial, which is 0 for an ideal gas. In order to validate our approach using the microscopic pressure (measurement in the data space), we can compare the result with the virial rule (the mathematical model).



**Figure 6. Pressure vs. time for ideal gas simulation**

Figure 6 shows the measured pressure vs. the time since the start of the experiment for a fairly large probing volume, and for a small probing volume. The rise in the area indicated with  $A$  shows when the user fully enters the gas with the probing volume. Note that after initial fluctuations, the pressure stabilizes around the value calculated from the virial theory (dashed line), as expected. Note also that for a smaller volume, fluctuations in the pressure are larger, as less particles are probed.

## 5.2. Study 2: Charge-Recombination Simulation

The charge recombination simulation calculates the effect of the entering of a high energetic particle into a liquid. The main result is the forming of ion-electron pairs

due to the collisions of the entering particle with the liquid-particles. Due to the electrostatic force, the electrons are attracted back to the ions, and will recombine to form the neutral gas particles. However, electrons that are too far away due to diffusion (see *escape radius* in Figure 8(b)), do not get attracted enough to make it back to the ions, and drift off. This process takes place in a very short time (picoseconds), and no tool exists to measure what exactly the electrons do in this short time frame. What can be measured, however, is the fraction of electrons that do not recombine with the ions.

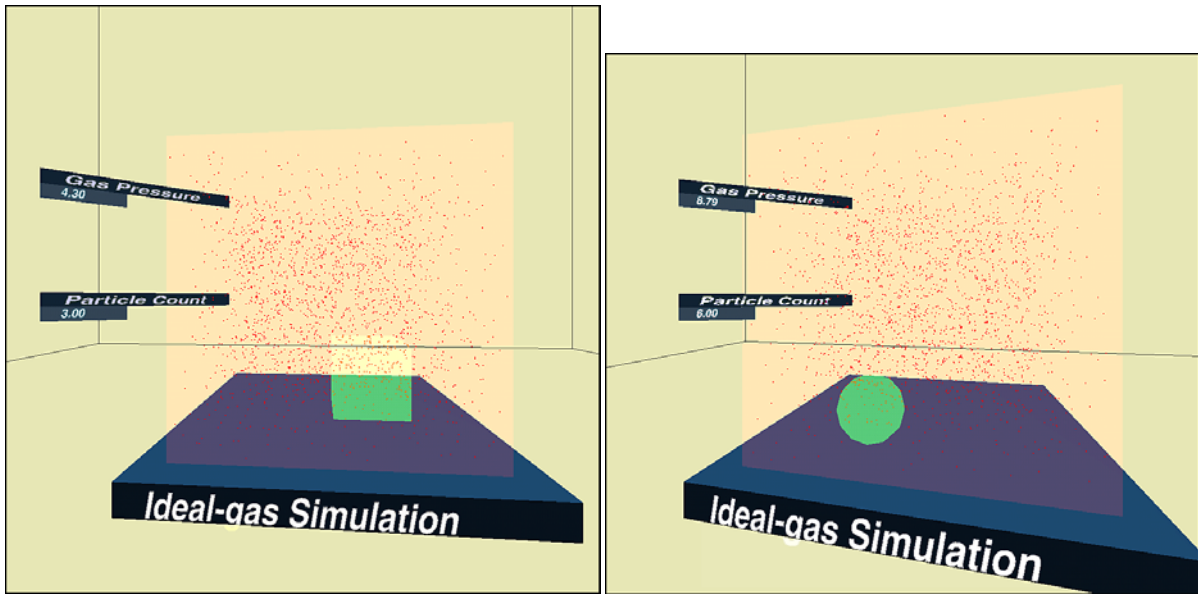
Simulating this phenomenon brings forward interesting new features. Most important, the time scale can be arbitrary, so the user has the ability to see non-averaged local effects of the recombination process. Furthermore, the user is able to interact closely with the process, by for instance, changing the initial position of an electron at the start of the simulation. All mentioned measurements are virtually impossible to perform for a real-world ion-recombination track.

Figure 8 shows the virtual environment for this simulation. The left figure shows the data space from the side where the user interacts with the particles. The right figure shows a top view of the data space. The central core of positive particles is shown by plus-symbols, and the cloud of electrons is shown by the squares. In the data space, the user can move and scale a cylinder, which measures particle flux through its surface. If the cylinder has a radius which is exactly the escape radius, no electrons will go in or out the cylinder during the simulation. If the cylinder is bigger, electrons will go outward due to the diffusion force, and if the cylinder is smaller, electrons will go inward, due to the attracting electrostatic force and finally recombine with the ions.

## 6. Conclusion and Future work

We sketched a new paradigm for steering and measuring within virtual reality environments. 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's input and output parameters. We addressed virtual measurement paradigms and special issues that arise like calibration and validation of the measuring tool.

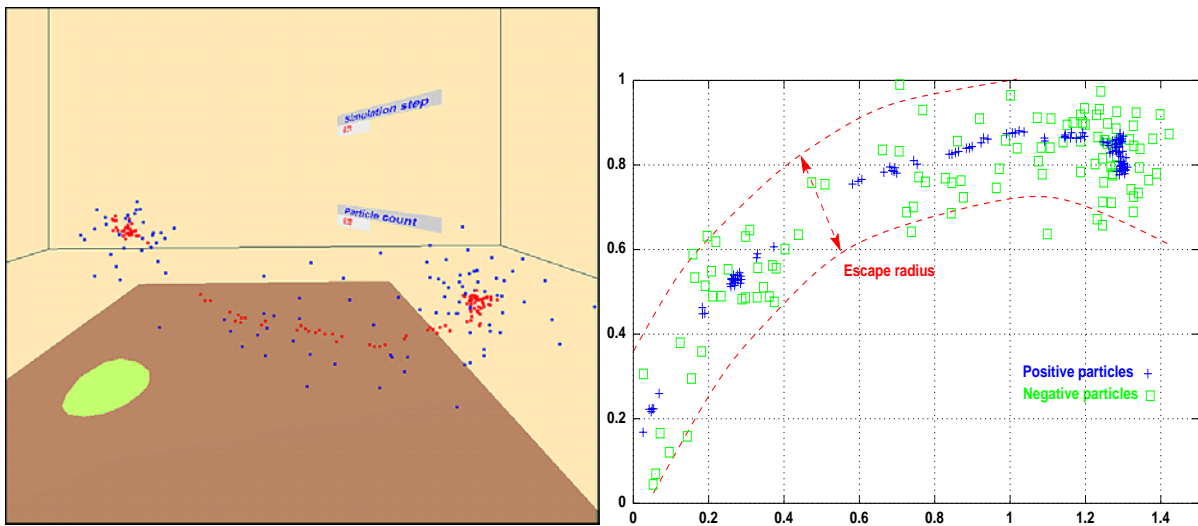
We connected two existing unmodified simulation programs to a VR environment, and showed that it is possible to measure from the visual domain. *CAVEStudy* allows to easily incorporate under-development applications to the measuring environment, using unmodified source code. Furthermore, the measuring functions (selection, counting, averaging, and derivatives) remain very general and applicable to a large class of applications. All functionalities together



(a) Cubic probe

(b) Spherical probe

**Figure 7. Gas simulation snapshots**



(a) VR view

(b) Top view (off-line)

**Figure 8. Charge-recombination simulation views**

result in a high-level steering environment. The scientist focuses only on the measuring tasks, relieved from graphic and coupling programming. We showed that applying measuring paradigms can contribute significantly to a scientist's understanding of his experimentation or simulation results. For the gas simulation, we showed that the measurement result can be validated and thereby trusted. The particle-recombination simulation opens a new way to efficiently interact and explore a non-observable phenomenon.

In our future work, we will provide a generic set of interaction, selection, and measuring functions. We will apply these on some new applications like molecular dynamics simulation and robot exploration.

## 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, Aug. 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, Feb. 1998.
- [3] C. Carlsson and O. Hagsand. DIVE — A Platform for Multi-User Virtual Environments. *Computers and Graphics*, 17(6):663–669, Nov.–Dec. 1993.
- [4] C. Cruz-Neira, D. Sandin, and T. DeFanti. Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE. In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 135–142, Aug. 1993.
- [5] A. Foster and C. Kesselman. *The Grid: Blueprint for a New Computer Infrastructure*. Morgan Kaufman, 1998.
- [6] 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.
- [7] C. Greenhalgh and S. Benford. MASSIVE: A collaborative virtual environment for teleconferencing. *ACM Transactions on Computer-Human Interaction*, 2(3):239–261, 1995.
- [8] A. S. Grimshaw and W. A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, Jan. 1997.
- [9] J. Hirschfelder, C. Curtiss, and R. Bird. *Molecular Theory of Gases and Liquids*. Wiley, New York, 1954.
- [10] G. G. II, J. Kohl, and P. 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. Jablonowski, J. Bruner, B. Bliss, and R. Haber. VASE: The Visualization and Application Steering Environment. In *Proceeding of Supercomputing '93*, pages 560–569, 1993.
- [12] J. Leigh, A. Johnson, T. DeFanti, and M. Brown. A Review of Tele-Immersive Applications in the CAVE Research Network. In *IEEE Virtual Reality '99*, pages 180–187, 1999.
- [13] 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.
- [14] U. Obeyesekere, F. Grinstein, and G. Patnaik. The Visual Interactive Desktop Laboratory. *IEEE Computational Science and Engineering*, 4(1):63–71, Jan. 97.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] D. Reed, Giles, and C. Catlett. Distributed Data and Immersive Collaboration. *Communication of the ACM*, 40(11), Nov. 1997.
- [19] 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, Pittsburgh, PA, Aug. 2000. IEEE Computer Society Press.
- [20] F. J. Seinstra, H. E. Bal, and H. J. Spoelder. Parallel Simulation of Ion Recombination in Nonpolar Liquids. *Future Generation Computer Systems*, 31(13):261–268, 1997.
- [21] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled Simulation in Virtual Reality with the MR Toolkit. *ACM Transactions on Information Systems*, 11(3):287–317, July 1993.
- [22] G. Singh, L. Serra, W. Png, and H. Ng. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence*, 3(1):19–34, 1994.
- [23] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.
- [24] H. J. Spoelder. Virtual Instrumentation and Virtual Environments. *IEEE Instrumentation and Measurement Magazine*, 3(3):14–19, 1998.
- [25] H. Tramberend. Avocado : A Distributed Virtual Reality Framework. In *IEEE Virtual Reality '99*, pages 14–21, 1999.
- [26] J. van Wijk and R. van Liere. *An Environment for Computational Steering*. Computer Society Press, 1997.
- [27] J. Vetter and K. Schwan. High Performance Computational Steering of Physical Simulations. In *Proceedings of the 11th IPPS '97*, pages 128–132, 1997.
- [28] 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.