Computational Steering

Robert van Liere

Centrum voor Wiskunde en Informatica
P.O. Box 94079, 1090 GB Amsterdam
The Netherlands
E-mail: robertl@cwi.nl

Jurriaan D. Mulder

Centrum voor Wiskunde en Informatica
P.O. Box 94079, 1090 GB Amsterdam
The Netherlands
E-mail: mullie@cwi.nl

Jarke J. van Wijk

Netherlands Energy Research Foundation ECN
P.O. Box 1, 1755 ZG Petten
The Netherlands

Centrum voor Wiskunde en Informatica
P.O. Box 94079, 1090 GB Amsterdam
The Netherlands

E-mail: vanwijk@ecn.nl

Abstract

The traditional cycle in simulation is to prepare input, execute a simulation, and to visualize the results as a post-processing step. However, more insight and a higher productivity can be achieved if these activities are done simultaneously. This is the underlying idea of *Computational Steering*: researchers change parameters of their simulation on the fly and immediately receive feedback on the effect.

In this paper the Computational Steering Environment, CSE , developed at CWI is described. We discuss the requirements of computational steering environment, its relation with high performance computing and networking, and show an application of its use.

CR Subject Classification (1991): I.3.4, I.3.6
Keywords & Phrases: computational steering, scientific visualization, three-dimensional graphics and interaction
Note: This paper was presented at the HPCN Europe '96, April 16-18, Brussel

1. Introduction

Scientific Visualization has been a research area since 1987, when the influential report of the US National Science Foundation was published [1]. Since then many new methods, techniques, and packages have been developed. Most of these developments are focussed on *post-processing* of data-sets. Usually the assumption is made that all data is generated first, after which the researcher iterates through the remaining steps of the visualization pipeline (selection, filtering, mapping, and

rendering) to achieve insight in the generated data. Hence, with post-processing the interaction with the simulation is limited.

Computational steering is a form of scientific visualization that is quite different than post-processing and can be considered as the ultimate goal of interactive computing. Computational steering enables the researcher to change parameters of the simulation while the simulation is in progress. As an example, Marshall of the Ohio Supercomputer Center has applied computational steering to the study of a 3D turbulence model of Lake Erie [2]. Their conclusions were: "Interaction with the computational model and the resulting graphics display is fundamental in scientific visualization. Steering enhances productivity by greatly reducing the time between changes to model parameters and the viewing of the results."

Computational steering has a strong relation with high performance computing and networking. First, to gain insight in the ever increasing complexity of high performance simulations, post processing falls short and more advanced methods, i.e. computaional steering, are needed. Second, high performance computing is needed to execute simulations and rendering at interactive speeds. High bandwidth and low latancy networks are needed to handle the amount of data produced by simulations interactively. If interactive speeds cannot be obtained, then computational steering will lose most of its value.

Computational steering is an attractive concept, but its implementation is cumbersome. A researcher must cooperate with a specialist in user-interfaces and visualization to develop a tool for the analysis of the output of the simulation. When the tool is ready, after some weeks or months, chances are high that interests of the researcher have shifted. Also, the further analysis of the data will introduce new research questions, which induce modifications of the tool. The close cooperation between researcher and the visualization specialist for an extended period is required. More appropriate would be to provide an environment in which researchers themselves can build interfaces and visualizations to the simulation. This would result in a more effective and efficient model – simulate – analysis cycle.

The CSE is a software environment for computational steering [3]. The CSE provides a collection of methods, techniques, and tools that enable researchers to apply computational steering. First, a number of requirements which we believe are fundamental for a steering environment are given. We then present the key concepts of the CSE 's architecture and the tools provided for the visualization of and interaction with the data. Finally, an application is discussed as an illustration of the use of the CSE .

2. REQUIREMENTS

Consider figure 1, which depicts the data-flow between a researcher and simulation via a CSE . A number of requirements for a steering environment can be given. First, the researcher enters new values for parameters, and views visualizations of the resulting data. Hence, input widgets such as text-fields, sliders, buttons, as well as a variety of visualization methods, such as graphs, text, graphics objects, etc. must be provided. Graphical objects must be provided that allow for two-way communication: both input and output. It must be possible to select and drag visualization objects, thereby directly controlling parameters and state variables of the simulation.

The simulation receives from the CSE new parameter values, and returns newly calculated results to the CSE . We assume that the simulation can handle changes of parameters on the fly, and that it can

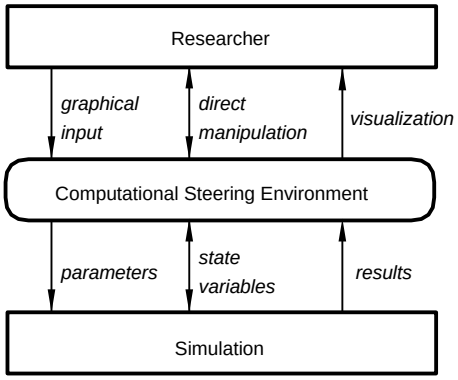

Figure 1: Data flow between researcher, CSE, and simulation.

provide meaningful intermediate results within a time-interval that is acceptable to the researcher.

The process of achieving insight via simulation is an incremental one. The researcher must be able to create and refine the interface to the simulation easily and incrementally. For all stages of the visualization pipeline (from simulation to rendering) the cycle *specification, implementation, application* is continuously reiterated.

The architecture of the CSE must be modular. There are two reasons for this: First, it must be possible to integrate existing tools, e.g., a special purpose package for grid-editing, in the CSE . Second, simulations usually execute on remote compute servers. Modular architectures simplify embedding simulations in the CSE .

The final requirement concerns the underlying data model and the amount of data movement within the CSE . The type of data to be handled depends very much on the type of simulation, and therefore can vary from simple scalar data to large, three-dimensional, time-dependent vector and tensor field data-sets. The underlying data model must be flexible enough to support a wide range of data types. Also, due to the quantity of data output from the simulation, the CSE must be able to handle large data sets efficiently.

3. Computational Steering Environment

3.1 Architecture

An overview of the architecture of the CSE is shown in figure 2. The architecture is centered around a *Data Manager* that acts as a blackboard for communicating values. Separate processes (*satellites*) can connect to the Data Manager and exchange data with it. The simulation is packaged as a satellite.

The purpose of the data manager is twofold. First, it manages a database of variables. Satellites can create, open, close, read, and write variables. For each variable the data manager stores a name, type, and value. Second, the data manager acts as an event notification manager. Satellites can subscribe to state changes in the data manager. When such a state change occurs the satellite will receive a notification from the data manager. For example, if a satellite subscribes to mutation events on a particular variable, the data manager will send a notification to that satellite whenever the value of the variable is mutated.

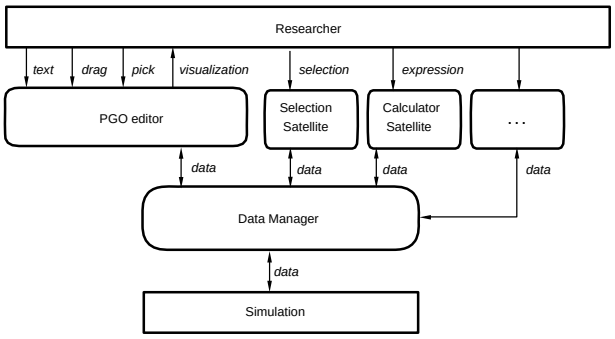

Figure 2: The CSE architecture

The kernel of the CSE architecture consists of the data manager and low level libraries that can access this functionality. The kernel is designed to be very simple, flexible and minimalistic. Unique in the CSE architecture is that higher level system functionality is pushed into satellites and not implemented in the kernel [4]. For example, a synchronization management satellite has been developed which allows visual specification of triggering and synchronization criteria between satellites. Another example of a system satellite is the transfer tuple satellite, which allows for efficient data transfer between two data managers.

Communication of a satellite with the Data Manager is done via a small Application Programmers Interface (API). The abstractions used are similar to standard UNIX I/O file handling, with variables instead of files. The functionality provided by this API is compact, terse and complete, but not simple to use. Therefore, on top of this interface a Data I/O library was defined, which is tuned to the needs of researchers that want to integrate their simulations within the CSE . The Data I/O library is simple to use and hides the complexities of the low level interface.

A large collection of general purpose visualization and data manipulation satellites have been built. Examples are satellites that implement data slicing, logging, calculation, transformation, and annotation.

3.2 Parameterized Graphics Objects

The most predominant satellite is the PGO editor, an interactive, MacDraw-like, graphics editing tool. There are two versions of the graphics editing tool, a 2D and a 3D version [5]. The central concept of the graphics editor is the Parametrized Graphics Object (PGO) : an interface is built up from graphics objects whose properties are functions of data in the data manager. The PGO editor has two modes: specification and application, or *edit* and *run*. In edit mode, the researcher can edit graphics objects and parameterize their geometry and attributes with variable names in the data manager. Hence, in edit mode, the researcher sketches a specification of the interface. In run mode, a two-way binding is established between the graphics objects and variable names in the data manager. Simulations may steer the interface by mutating the data bound to the graphics objects. Similarly, researchers drive the simulation by manipulating graphics objects. Hence, in run mode, a two-way communication between graphics and data in the simulation is realized.

As a simple example of how one would use the PGO editor, consider the left side of figure 3, which depicts the specification of an arrow. The right side shows that application of the arrow, after being

bound to an array of values in the data manager. The arrow could for instance be used to steer a field force in the simulation. Its length would then be parametrized to the magnitude of the force while its orientation would depict the direction of the force.

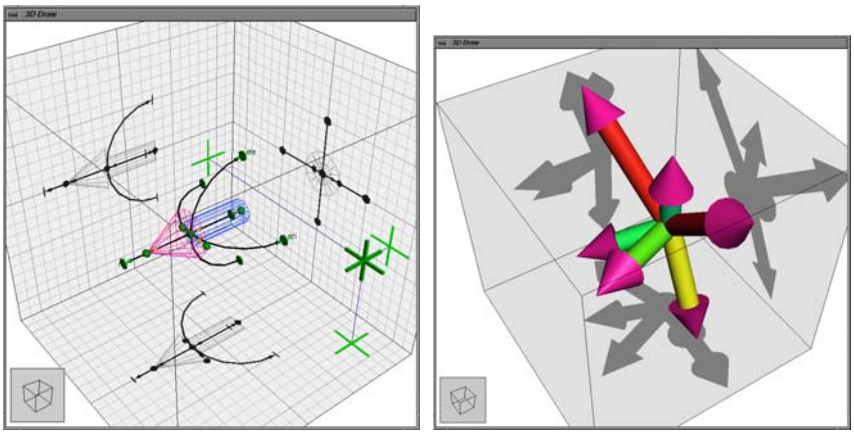

Figure 3: Arrow in edit mode (left) and run mode (right).

Libraries of predefined widgets which are customized towards specific application needs can be made available. In addition, the 3D PGO editor provides techniques that simplify 3D interaction with graphics objects. Two examples are the 3D crosshair cursor and shadow editing. The crosshair cursor provides the user with fine control over the positioning and translating of points and objects in 3D space. Shadow editing allows the user to interact with the orthogonal projections (shadows) of the objects on a bounding box. An example of shadow editing is shown in figure 3.

4. Atmospheric Simulation Application

The CSE has been applied to the simulation of a model for smog prediction over Europe [1]. The full blown model forecasts the levels of air pollution, which is characterized by approximately 104 reactions between ca. 70 species. For example, the concentrations of ozone (O_3), sulphur dioxide (SO_2) and sulphate aerosol (SO_4) are calculated. The vertical stratification is modeled by four layers; the surface layer, the mixing layer, the reservoir layer, and the upper layer. The computational model is described by a set of partial differential equations that model advection, diffusion, emission, wet and dry deposition, fumigation, and chemical reactions.

An important numerical utility to solve these equations is local grid refinement [6]. This technique is used to improve the quality of the model calculations in areas with large spatial gradients (for example in regions with strong emissions). The trade off to be made in local grid refinement is calculation accuracy versus computation speed. The CSE has been used to steer various aspects of the smog prediction simulation:

- Control of the tolerance value that determines where refinement is necessary;
- Editing of emission data;

[1] Thanks to M. van Loon and J.G. Verwer of the Afdeling Numerieke Wiskunde, CWI for providing all information and code of this application.

- Use of a bounding box as a concentration probe; The coordinates of the bounding box steer the slicing satellite, which in turn triggers the calculator and logging satellites. The result of the logging satellite triggers the PGO editor;
- Interactive control over simulation time.

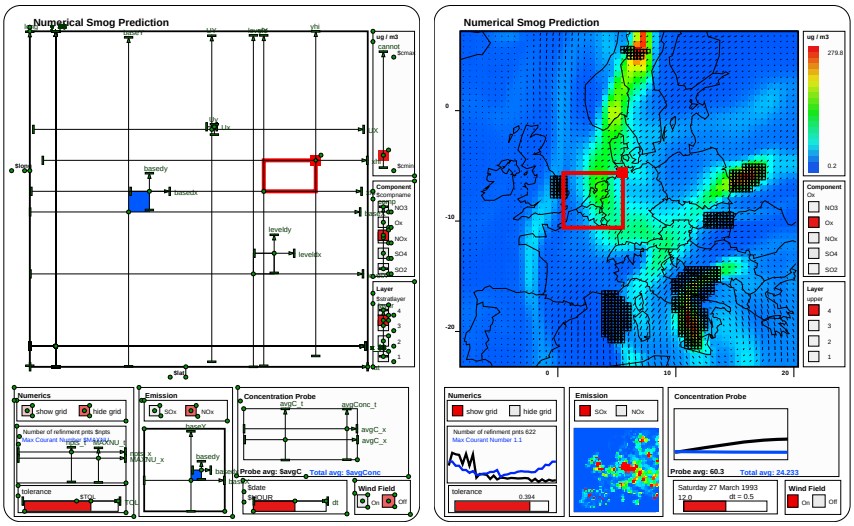

Figure 4: Smog prediction simulation: edit mode (left) and run mode (right)

The left of figure 4 shows a snapshot of the PGO editor in edit mode, the right shows a snapshot in run mode. The concentration of ozone in the upper layer is shown in color along with areas of local grid refinement (shown as smaller rectangles). The wind field is shown as small black vectors.

This set-up enables a numerical mathematician to gain insight in the relationship between grid refinement tolerance, the maximal Courant number, and the simulation time. Sliders control the grid refinement tolerance and simulation time. The graph on the lower left shows a log of the number of grid cells that are refined and the maximum Courant number. The *dmlog* satellite records the data for display. The graph immediately displays the effects of changes on the tolerance or the simulation time.

This particular configuration runs at approximately five frames a second on a modern workstation. The amount of data involved is substantial: depending on tolerance level, the amount of data may vary between one and four megabytes per time step. The simulation has 447 time steps. Approximately 90 percent of the CPU time was taken by the simulation. The remaining 10 percent was used by the other satellites and data transport in the CSE .

5. Discussion

In this paper we presented the requirements and design of the CSE , an environment for computational steering. The design of the CSE 's architecture was driven by the following basic concepts:

- The use of *low-level primitives*: a simple data model and graphics objects. The interfaces to these primitives are familiar to the end-user: a simple I/O library for data manipulation and the

PGO editor for graphics.

- *No higher level semantics* are defined in the kernel. As a result, the environment is general and flexible. High level features are built on top of the kernel by putting this functionality into satellites. By pushing high level functionality into satellites, the CSE provides an environment that is extensible and reuseable.

- The data manager, the PGO editor and all other satellites rely on *late binding* of named variables. As a result, it is possible to iteratively define new visualizations or define different bindings to output data.

- All operations in the data manager and satellites are based entirely on *data*. For example, in the PGO editor, dragging, picking and text input are translated into changes of data. The predominant type of event within the CSE is the data mutation.

The CSE currently runs on SGI, Sun, DEC Alpha, HP, Cray YMP and IBM SP1 platforms. The CSE uses the device independent graphics package OpenGL for the 3D and X11 for 2D version of the PGO. It uses TCP/IP for transport on Ethernet or ATM networks.

As computational models become more complex, researchers have an increasing need for interactive tools in which computational models can be explored in more effective ways. From our experience we believe that CSE offers a powerful, though basic, platform for computational steering of high performance simulations.

REFERENCES

1. B. McCormick, T. Defanti, and M. Brown. Visualization in Scientific Computing. *Computer Graphics (SIGGRAPH '88)*, 22(6):103–111, 1987.
2. R.E. Marshall, J.L. Kempf, D. Scott Dyer, and C-C Yen. Visualization Methods and Simulation Steering a 3D Turbulence Model of Lake Erie. *1990 Symp. on Interactive 3D Graphics, Computer Graphics*, 24(2):89–97, 1990.
3. J.J. van Wijk and R. van Liere. An Environment for Computational Steering. Technical Report CS-R9448, Centre for Mathematics and Computer Science (CWI), 1994. Presented at the Dagstuhl Seminar on Scientific Visualization, 23-27 May 1994, Germany, proceedings to be published.
4. R. van Liere and J.J. van Wijk. CSE : A Modular Environment for Computational Steering. Technical report, Centre for Mathematics and Computer Science (CWI), 1995. publication pending.
5. J. Mulder and J.J. van Wijk. 3D Computational Steering with Parameterized Geometric Objects. In *Proceedings Visualization '95*, pages 304–311. IEEE Computer Society Press, Los Alamitos, CA, 1995.
6. M. van Loon. Numerical smog prediction: Grid refinement. Technical Report NW-R9448, Centre for Mathematics and Computer Science (CWI), 1994.