

Computational Steering

Robert van Liere,^{a,1} Jurriaan D. Mulder^a Jarke J. van Wijk^{a,b}

^a *Center for Mathematics and Computer Science CWI, P.O. Box 94097, 1090 GB Amsterdam, Netherlands*

^b *Netherlands Energy Research Foundation ECN, P.O. Box 1, 1755 ZG Petten, Netherlands*

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, 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.

Keywords: scientific visualization; computational steering; three-dimensional graphics and interaction.

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 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. It enables the researcher to change parameters

¹ Partially supported by the Dutch ICES-HPCN programme

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."

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. More advanced interactive visualization methods are needed. Second, high performance computing is needed to execute simulations and rendering at interactive speeds. High bandwidth and low latency networks are needed to interactively handle the vast amount of data produced by HPC simulations. If interactive speeds cannot be obtained, then most of the merits of computational steering will be lost.

Computational steering is an attractive concept, but its implementation is cumbersome and time consuming. 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. The format of this paper is as follows: First, a number of requirements which we believe are fundamental for a steering environment are given. We then present some key concepts of the CSE 's architecture and the tools provided for the visualization of and interaction with the data. Finally, two applications are 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 visual-

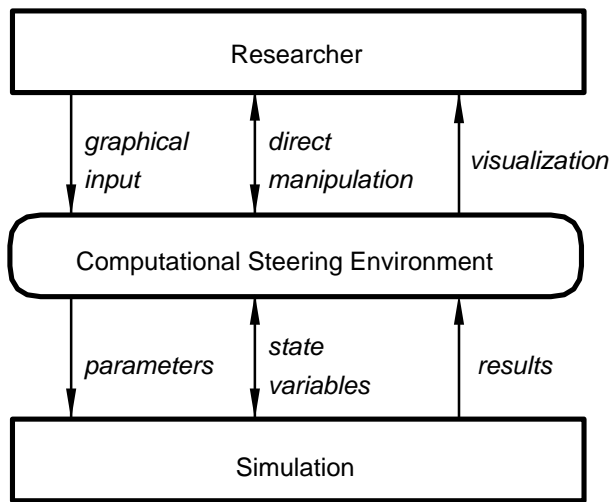


Fig. 1. Data flow between researcher, CSE, and simulation.

izations 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 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

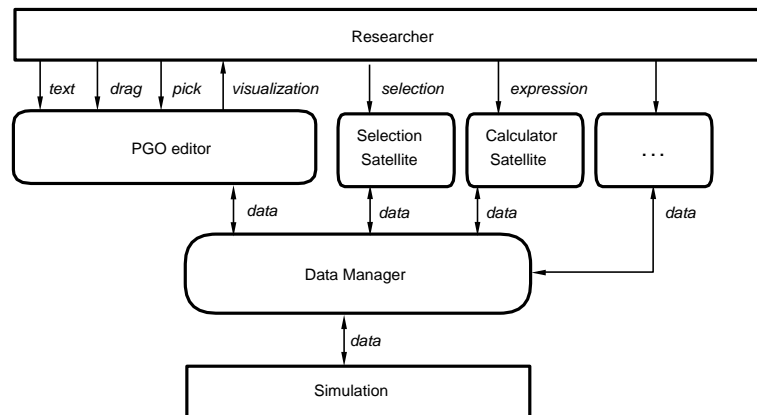


Fig. 2. The CSE architecture

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, when 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.

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.

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 *Integration of simulations*

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.

What were the design requirements for the Data I/O library ? By far the most important is that the required changes to the simulation code are absolutely minimal. The researcher will not accept to rewrite the (FORTRAN) application; e.g. by changing the control structure from straightforward iteration into an event-driven structure. Additional bookkeeping should not be necessary. In other words, the researcher must be provided with only a few simple routines, just to declare and communicate variables. We present the Data I/O library, with a simple but generic example. Though this particular example uses C-language bindings, Fortran-language bindings are supported as well.

```
simulation(float *s, float *t, int *size, float *x)
{
    int continue = TRUE;

    /* Open connection, connect and subscribe variables */

    dioOpen("borneo.cwi.nl");
    dioConnectFloat("s", s, READ);
    dioConnectInt("continue", &continue, READ);
    dioConnectFloatArray("x", x, 1, size, UPDATE);
    dioConnectFloat("t", t, WRITE);

    /* simulation loop and update data */

    while (continue)
    {
        t = t + 1.0;
```

```

        calculate_values(t, s, size, x);
        dioUpdate();
    }
    dioClose();
}

```

The structure of this example, which is typical for a continuous simulations, consists of two parts. First, variables are initialized. The required changes to an existing source code would be limited to opening and closing a connection with the Data Manager and connection of the variables via the `dioConnect` routines. Second, a main loop is entered where time is incremented and new values are calculated. The required changes to the source code is a single call to exchange data. The locations where to insert these calls are easy to find: typically at the outer level of the simulation program.

The first parameters of the `dioConnect` routines are the name of the variable and its address. For the connection of arrays the number of dimensions and their sizes must also be specified. The last parameter describes the direction of the data flow. This information is used by the `dioUpdate()` routine to decide what must be done. In `dioUpdate()` first the event stream from the Data Manager is checked if variables to be read or updated have changed. If so, these variables are read from the Data Manager. Next the values of all unchanged variables are written to the Data Manager. The net result of `dioUpdate()` is that all connected variables have the same value in the simulation and Data Manager. With these few calls the user can steer parameters (`s`) of the simulation, to stop the simulation (`continue`), monitor its progress (`t`, `x`) or even to change state variables (`x`).

The simulation in the example executes asynchronously, without waiting for external events. After each iteration all data is read or written. Alternatively, the routine `dioSetSyncVar()` can be used if the simulation must execute synchronously with other satellites.

To deal with more subtle situations variables can be grouped into sets. In the main loop the application can read and write specific sets, and wait until a particular set changes. Hence, a more efficient use of resources can be realized with a small additional effort.

3.3 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 graph-

ics 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. In addition

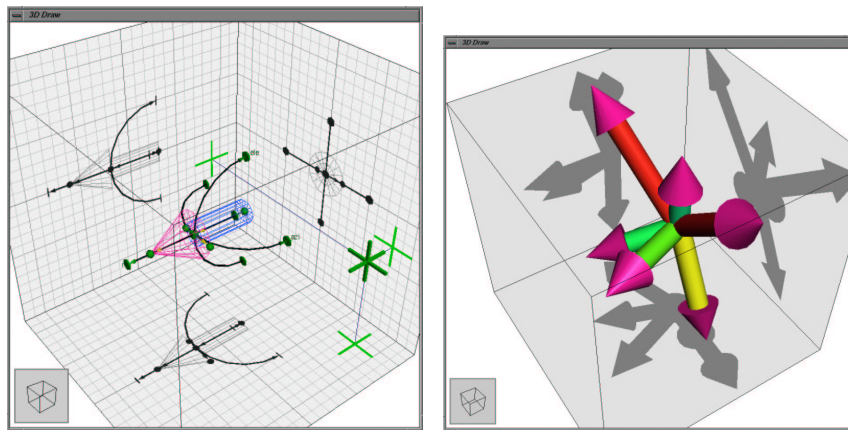


Fig. 3. Arrow in edit mode (left) and run mode (right).

to facilities for graphical object specification, the 3D PGO editor provides techniques that simplify interaction with objects. Every graphical object that is bound to data may be directly manipulated. Two other 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.

The combination of the Data I/O library and the PGO editor has led to a very interactive and iterative way of gaining insight to a simulation. We call this type of working method “what you draw is what you control”, and is characterized by the following loop:

- (i) Decide which parameters are important for control and visualization;

- (ii) Use the Input/Output library to connect parameters;
- (iii) Use the PGO editor to edit an interface;
- (iv) Run the interface: view and control the simulation;
- (v) Analyze the results and go back to one of the previous steps.

4 Applications

4.1 Atmospheric Simulation

The CSE has been applied to the simulation of a model for smog prediction over Europe ². The full blown model forecasts the levels of air pollution, which is characterized by approximately 25 reactions between ca. 20 species. For example, the concentrations of ozone (O₃), sulphur dioxide (SO₂) and sulphate aerosol (SO₄) 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 grid refinement is necessary;
- Editing of emission data;
- 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.

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

²Thanks to M. van Loon and J.G. Verwer of the Afdeling Numerieke Wiskunde, CWI for providing all information and code of this application.

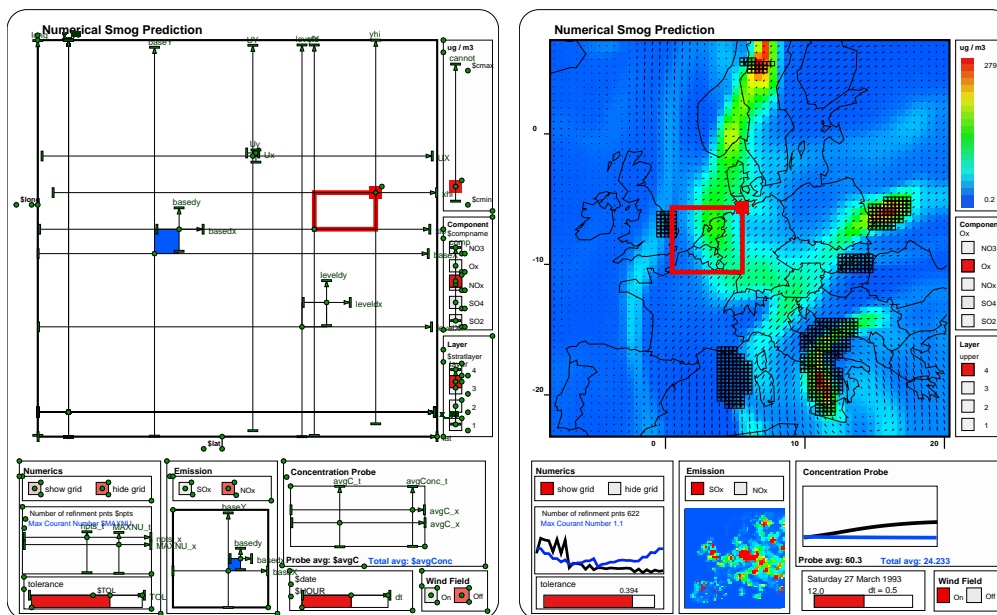


Fig. 4. Smog prediction simulation: edit mode (left) and run mode (right)

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.

Figure 5 is a snapshot of the satellite configuration of the smog prediction model. The configuration consists of eight uniquely labeled satellites. The nodes of the graph represent the satellites connected to the data manager. The blue edges indicate data flow dependency. A blue arrow indicates a directed data dependency. Green edges indicate a trigger graph; a graph that defines the synchronous execution order of satellites. The panel on the top right provides additional variable information of a selected satellite. The panel on the bottom is the interface to the trigger graph editor. 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 .

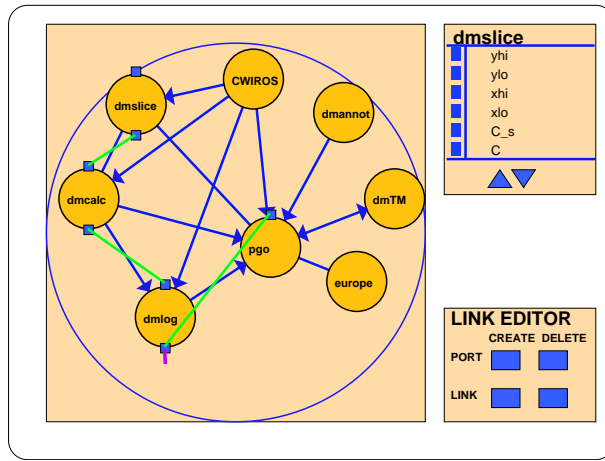


Fig. 5. Satellite configuration of the smog prediction model.

4.2 Computational Fluid Dynamics

A classical computational fluid dynamics problem is the analysis of the flow field around an airfoil under various far-field conditions.³ Given a number of assumptions, this problem can be mathematically modeled by the so-called Euler equations of gas dynamics. There exist a number of numerical methods to solve discretized forms of the Euler equations. One class of such methods uses adaptive multi-grid techniques. At CWI multi-grid methods are studied intensively, especially for the solution of the compressible Euler equations [7]. The efficiency of adaptive multi-grid methods for the solution of systems of partial differential equations is superior to that of other solution methods.

The problem studied here is the flow around the well-known NACA0012-airfoil. The user can change the angle of attack and the Mach number, the solver calculates the corresponding pressure, density and velocity fields. Small perturbations of these parameters give insight in various aspects of the flow field. Since new parameter values define a completely new problem, the solver is reinitialized each time a parameter is changed. Intermediate results of the solver are displayed to show the convergence of the process.

Figure 6 shows an PGO interface to the multi-grid flow solver. On the left is the PGO interface in edit mode. To the right is the PGO interface in run mode. The upper left and upper right panels show the pressure and density fields. The adaptive grid refinement is shown in the lower right panel. Various parameters that control the adaptive grid refinement scheme, such as the refinement tolerance and the maximum depth of the refinement process, can be changed on the fly. Hence, the user can make a trade off between fast, but

³ Thanks to E. van de Marel and B. Koren of the Afdeling Numerieke Wiskunde, CWI for providing all information and code of this application.

possibly inaccurate results, and slow, precise results. The time needed to de-

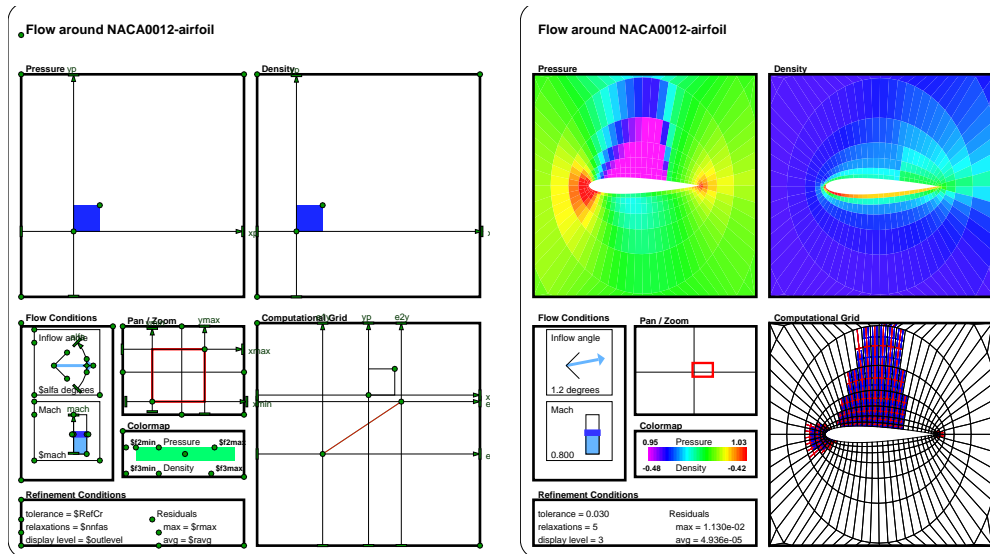


Fig. 6. Multi-grid solution of Euler equations : edit and run mode.

velop this interface was about three afternoons. Alternative visual interfaces to the simulation can now be defined interactively during analysis sessions.

The total simulation time is about 2 seconds for a simulation on a grid of size 32×128 for the lowest level and 128×512 for the highest level. Intermediate results are displayed instantaneously. The simulation runs on a four CPU SGI Challenge and the visualization is performed on an SGI Indigo2 High Impact workstation. The systems are connected through an ATM network.

5 Related Work

Many research and development teams have designed and implemented interactive visualization environments. Williams, Rashure and Hanson [8] provide a framework to understand design tradeoffs when developing data flow based visualization systems. Gu, Vetter and Schwan [9] give a comprehensive annotated bibliography of many aspects of interactive program steering.

Giving an in depth analysis of other visualization environments is beyond the scope of this paper. Instead, we focus only on some issues that are found the CSE :

- In data flow environments, visualization operators are combined by linking output and input ports. Operators are executed upon availability of data on the input port. Operators are packaged as modules, and most data flow environments provide high level tools for building modules.

IRIS Explorer [10] is an example of a data flow visualization environment. There are many fundamental differences between IRIS Explorer and CSE . First, direct manipulation is very difficult to achieve in data flow environments. In IRIS Explorer there is no one-to-one relation between geometry and a corresponding object in an upstream module. This makes direct manipulation of objects in the simulation very tedious. In contrast, with CSE 's binding mechanism direct manipulation is ensured. Second, IRIS Explorer's mechanism to manage data transport differs from CSE 's. Data producing operators push data to downstream consuming operators. This may result in redundant data movement, if the downstream operator does not need the data. In contrast, a satellite in the CSE will only read data when it ready to consume it.

- Glyphmaker [11] is a system that allows users to customize graphical representations using a glyph editor and a simple point-to-click binding mechanism. Glyphmaker is implemented as a collection of modules in IRIS Explorer.

Allowing users to specify customized graphical representations resembles the main idea of the PGO editor, although the implementations are very different. Glyphmaker's graphics primitives are targeted to glyph specifications for the visualization of data, whereas the PGO 's graphics primitive set is larger, and aims at both visualization and user input.

- VIEW [12] is a system that is based on a tight coupling of on-screen geometry with a database. A data drawing tool allows users to define composite geometric objects by selecting primitive graphical components from the database. In addition, an event-definition mechanism allows the user to customize interaction sequences. A tool scripting language is used to specify these interaction sequences, and simple selection functions are offered to bind names in the scripts to geometry in the database. Event monitors are used to execute scripts.

A principle difference between VIEW and CSE is event handling. VIEW provides event monitors to customize interaction sequences. Events in VIEW include changes in input device state and picking of geometry. CSE notion of events is based exclusively on state changes within the data manager. Satellites may receive events by subscribing on the state change.

6 Discussion

As computational models become more complex, end users have an increasing need for interactive tools in which their computational models can be explored. Computational steering is useful in several areas. The standard application of computational steering is parameter variation such that the end user achieves insight. Technical discussions progress much faster if "What if?" questions can

be answered immediately. In section 4 we have shown examples of parameter variation. However, steering can also be applied to other areas. For example, steering can be used in model development, when initial model settings are sought. Also, in code development, steering can be used to debug numerical code.

In this paper we discussed the requirements and design of the CSE . An architecture with a central Data Manager gives flexibility and modularity. The Data Input / Output library was presented as an easy to use layer for data communication between Data Manager and simulation. With parametrized graphics objects end-users themselves can define customized visualizations to their data. The CSE thus enables end users to develop visualizations interactively, together with the development of their model and simulation. It is easy to connect new parameters and to define their visualization if new parameters have to be controlled or other results must be visualized.

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 C90 and IBM SP1 platforms. The CSE uses the device independent graphics package OpenGL for the implementation of the PGO and TCP/IP for data transport over Ethernet or ATM networks.

In our future work we aim at further improvement and expansion of the CSE . The conceptual model has proven to be simple and effective for users, hence these enhancements must be hidden from the end users. First, the resource management can be improved. All data sets are now routed via TCP/IP connections to the Data Manager. For small data sets this works well, for very

large data sets more efficient techniques are required. In particular, a centralized data manager is not efficient for the data movement of large data sets. Second, the current CSE provides only support for direct steering. Additional tools that support the user with automatic navigation in parameter spaces will be developed. Third, we will further explore novel presentation techniques, especially for 3D simulations. PGOs provide an excellent means for fast implementation of icons, composite 3D interactors and multiple scalar fields.

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. In M. Gobel, J. David, P. Slavik, and J.J. van Wijk, editors, *Proceedings of the 7th Eurographics Workshop on Visualization in Scientific Computing*, pages 256–266, Prague, April 1996. Springer-Verlag.
- [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 Methods in Smog Prediction*. PhD thesis, University of Amsterdam, June 1996.
- [7] P.W. Hemker, B. Koren, W.M. Lioen, M. Nool, and H.T.M. van der Maarel. Multigrid for steady gas dynamics problems. In H. Deconinck and B. Koren, editors, *Euler and Navier-Stokes Solvers Using Multi-Dimensional Upwind Schemes and Multigrid Acceleration*. Vieweg, Braunschweig, 1996.
- [8] C. Williams, J. Rasure, and C. Hansen. The State of the Art of Visual Languages for Visualization. In *Proceedings Visualization '92*, pages 202–209, 1992.
- [9] W. Gu, J. Vetter, and K. Schwan. An Annotated Bibliography of Interactive Program Steering. Technical Report GIT-CC-94-15, Georgia Institute of Technology, Atlanta, Georgia, 1994.

- [10] Explorer Development Team. Iris Explorer 2.0 Module Writer's Guide. Technical Report 007-1369-020, Silicon Graphics Inc, 1993.
- [11] W. Ribarsky, E. Ayers, J. Eble, and S. Mukherjea. Glyphmaker: Creating Customized Visualization of Complex Data. *IEEE Computer Graphics and Applications*, 27(4):57–64, July 1994.
- [12] L. Bergman, J. Richardson, D. Richardson, and F. Brooks Jr. VIEW – An Exploratory Molecular Visualization System with User-Definable Interaction Sequences. *Computer Graphics*, 27(6 (SIGGRAPH '93)):117–126, 1993.