# Logging in a Computational Steering Environment

Jurriaan D. Mulder
*CWI*
*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
mullie@cwi.nl

Jarke J. van Wijk
*ECN*
*P.O. Box 1, 1755 ZG Petten, The Netherlands*
and
*CWI*
*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
vanwijk@ecn.nl

**Abstract**

Logging of input and output variables is very useful in computational steering. In this paper we describe how we added logging functionality to a computational steering environment developed at CWI. We show how a 2D interface can be augmented with logging by using the third dimension for the display of the logged variables. The user specifies which graphical representations of variables must be logged, and this log is displayed together with the current state of the simulation. Two examples show that logging in computational steering gives more insight in the simulation, that it can be used for monitoring, and that it can be used to undo erroneous actions.

*CR Subject Classification (1991):* I.3.4, I.3.6
*Keywords & Phrases:* computational steering, scientific visualization, logging
*Note:* This paper was presented at the 1995 EuroGraphics Workshop on Visualization in Scientific Computing, May 3–5, Chia, Italy

# 1 Introduction

## 1.1 Computational Steering

Many new methods, techniques, and packages have been developed for scientific visualization in recent years. Most of these developments however, are limited to post-processing of data-sets and thus allow no interaction with the simulation itself. Two alternatives to this post-processing approach can be distinguished: *tracking* and *steering* [MKDY90, MH90]. With tracking, visualization is performed during simulation and the only interaction possible is to stop the simulation. With computational steering, simulation parameters can be altered in the ongoing simulation while viewing the results. Such interactive control of a computational model during execution allows the user to quickly discover and correct erroneous input parameter values, but more important, the user gains more insight if he can immediately observe the effect of changes in input parameters to dependent variables.

In [vWvL94] a general and flexible environment for computational steering is described, which has been developed at CWI. Within this environment, the user can easily develop user interfaces and 2D visualizations of his simulation.

## 1.2 Logging

Marshall et al. suggest several useful functions for steering applications [MKDY90], one of which is to keep a history of parameter settings. We agree that logging of parameters can be extremely useful in computational

steering, especially if such a log can be visualized together with the simulation. In addition, such logging should not be restricted to input variables, but also output variables should be logged. The simulation can take care of the logging, but a more generic and flexible approach is to integrate logging in an environment for computational steering. If a variable can be displayed, it should be possible to log that variable with little effort. As a result, not only the current state of the simulation is visualized, but also an overview of the history can be shown.

A graphical representation of the history of a simulation is useful in several ways. First of all, it provides the user with a better insight in relations between displayed variables. To see how a simulation has developed through time, at one glance or by browsing through its history, can be helpful to understand the results.

Secondly, it dismisses the user of constantly having to watch the simulation progressing. While doing other tasks, the user can check the results occasionally by looking at the visualization of the history of the simulation.

Thirdly, if the history is stored, more functionality can be provided than just its visualization. With a restore option, the user can jump back to the point of time where a special situation occurred and continue the simulation from there, steering it into a different direction. Such a restore option can also be used as an undo function.

Brodlie et al. emphasize the importance of history recording in a problem solving environment that integrates computation and visualization [BB93]. They developed the *history tree* concept, which reflects the history of the search process used by a scientist in reaching an optimal solution to a simulation. This tree is presented to the scientist, who can then perform operations on the tree such as visualize or restore previous simulation data and parameters. Our scope is different from that of Brodlie et al.. Our aim is not to show and handle the history of the process, but the history of the data.

One of the components of the computational steering environment developed at CWI is a graphics tool for the visualization of and interaction with the simulation. In this paper we describe how we extended this tool with logging functionality. In section 2 we give an overview of the computational steering environment. In section 3 we describe the new implementation of logging. Examples are presented in section 4, followed by the conclusions in section 5.

# 2 The Computational Steering Environment

The computational steering environment (CSE) developed at CWI consists of two major components. A Data Manager takes care of centralized data storage and event notification, and a graphics tool is provided to define a user interface interactively and to show 2D visualizations of the animation. The central concept here is the use of Parametrized Graphics Objects (PGOs): an interface is built up from graphics objects which properties are functions of data in the Data Manager. The architecture of the CSE is shown in figure 1.

## 2.1 The Data Manager

The central process in the CSE is the Data Manager. Other processes (called *satellites*) such as the simulation and the PGO editor, can connect to and communicate with the Data Manager. The purpose of the Data Manager is twofold. A database of variables is managed, and it takes care of event notification. Satellites can create and read/write variables, and they can subscribe to events, such as notification of mutations of a particular variable. Thus, the Data Manager enables the different satellites to use the same data and to communicate with other satellites.

## 2.2 The PGO Editor

The PGO editor is a tool for the graphical interaction with a simulation. It handles the visualization of the data, the user input, and direct manipulation. The editor has two modes: specification (*edit*) and application (*run*). In edit-mode, the user can create and edit graphics objects much like in MacDraw-like drawing editors. The standard objects offered are: fill-area, polyline, rectangle, circle, arc, and text. The properties of these

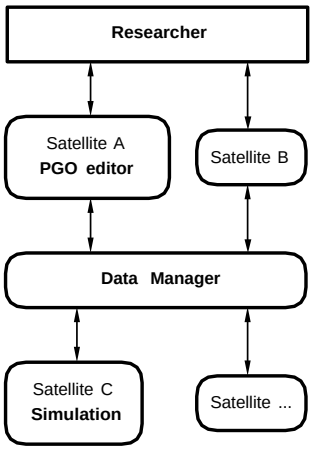


Figure 1: Architecture of the Computational Steering Environment.

objects (such as position and color) can be parametrized to values of variables in the Data Manager. Hence, the user draws a specification of the interface. In run-mode, a two way communication is established between the user and the simulation by binding these properties to variables. Data is retrieved from the Data Manager and mapped onto the properties of the graphics objects. The user can enter text, drag and pick objects, which input is translated into changes of the values of variables.

### 2.3 Logging Satellite

In [vWvL94] is described how logging can be accomplished with an additional satellite. This satellite keeps track of the changes of one or more variables present in the Data Manager and creates new variables in the Data Manager to store the logged values. With this approach, the logged variables are treated similar as the other variables in the Data Manager. The user must explicitly specify how these logged variables are visualized, in the same manner as the other variables, using the PGO editor.

## 3 Logging in the PGO editor

The approach we present here is to integrate logging in the PGO editor. The user can select graphics objects and specify that these objects should be logged. The PGO editor then, during run-time, automatically logs the associated variables and displays the logged graphics objects.

Because all graphics in the PGO editor is two-dimensional, we could map the time dimension on the third geometric dimension; the 2D display of the current state is projected on the front plane of a 3D box and the log is displayed inside this box. This way, the log can easily be displayed together with the current state, see figure 2.

The logged graphical objects can be displayed in two ways: as discrete slices and/or connected in between the slices. With the first display method, the graphical objects are shown in the time slices in the same way they were displayed in the front plane. In the second method, the logged graphical objects are connected in between consecutive slices. Thus, points become lines, lines become surfaces, and areas become volumes. In other words, the 2D graphics objects are swept along the time-axis, just like in translational sweeping in solid modeling [FvDFH90].

When visualizing an ongoing simulation, the user can pause the simulation and thereby freeze the display. The user can then browse through the logged history of the simulation. He can move a plane, parallel to the front plane, along the time-axis. All graphics in front of this plane is clipped. In addition, a restore function is

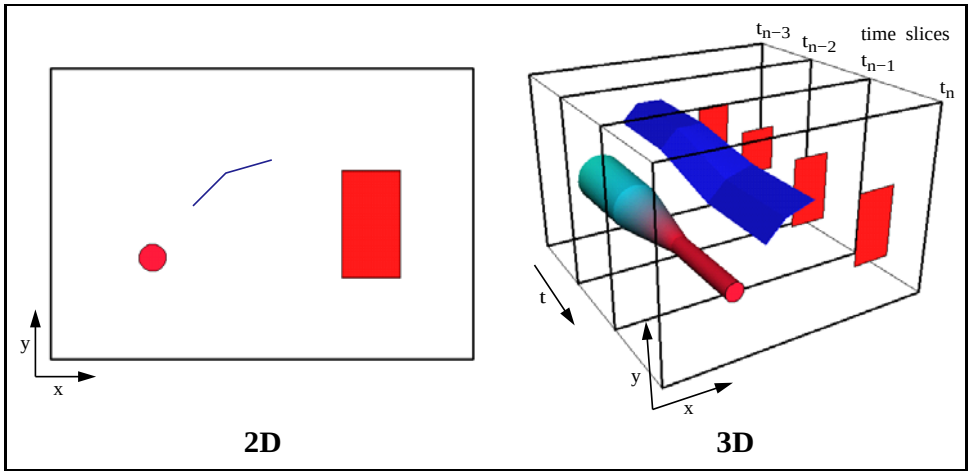


Figure 2: 2D display of current state and 3D display with time slices.

provided. The values of all input variables that are logged are retrieved for the selected time slice and restored into the data manager.

The user can further adjust his view on the box, the number of time steps to be logged, and the logging rate per object.

# 4 Examples

## 4.1 Sorting

This first example shows the use of the logging function for the visualization of two sorting algorithms: bubble sort and selection sort. Although our system is not intended for pure algorithm visualization, this example does show how a graphical representation of a log can be useful to understand an ongoing computation.

There are two arrays present in the simulation, each of which is sorted by one of the algorithms. The elements in the arrays can be rearranged into a random or decreasing order. Figure 3 shows the specification of the visualization of the two sorting algorithms. This specification results in a visualization of the two arrays present in the simulation. The elements of the arrays are represented by small rectangles parametrized on the represented value (color) and position in the array (screen position). Just below each rectangle a number is displayed to show the actual value of the element. The number of rectangles that will be drawn in run-mode equals the size of the arrays in the simulation. Two flags indicate whether the arrays are sorted or not.

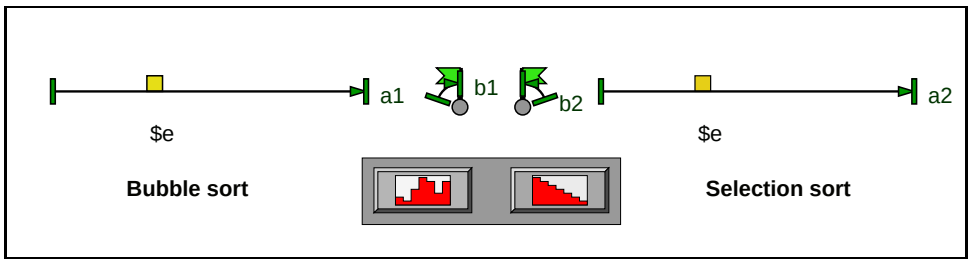


Figure 3: Visualization of sorting process, edit mode.

In addition, two buttons are specified which are used to rearrange the elements in the arrays. If one of these buttons is pressed, the simulation arranges the elements in the arrays to the desired order and starts sorting them. At each exchange in the arrays, the data in the Data Manager is updated, which in turn triggers the PGO editor to update the visualization.

When the sorting process is visualized without any logging, as shown in figure 4, it is not possible to get a good impression of the two sorting algorithms; all there is to see is a number of rapid changes in the arrays until they are sorted. However, if the graphical representation of the arrays is logged, it becomes clear how both sorting algorithms progress. From the displayed log (figure 5) it can easily be derived which elements were exchanged and in what order. Not surprising, the bubble sort algorithm uses far more exchanges than the selection sort.

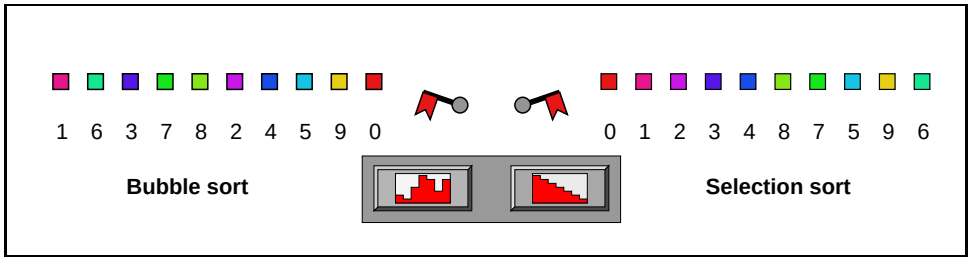


Figure 4: Visualization of sorting process, no logging.

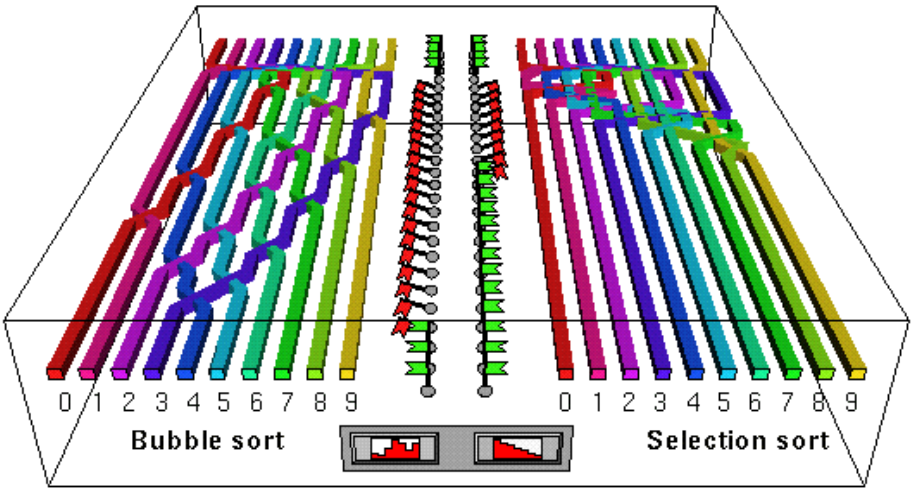


Figure 5: Logged visualization of sorting process.

## 4.2 Bouncing Balls

Another example is shown in figure 6, where the log of a simulation of bouncing balls is visualized. The balls are subject to a field force, a damping force of the medium, and contact damping in case of collisions. The color of the balls is parametrized to their velocity. The current kinetic energy of the system and the current

field force are visualized alongside the field. The user can control the field force and the positions of the balls through direct manipulation.

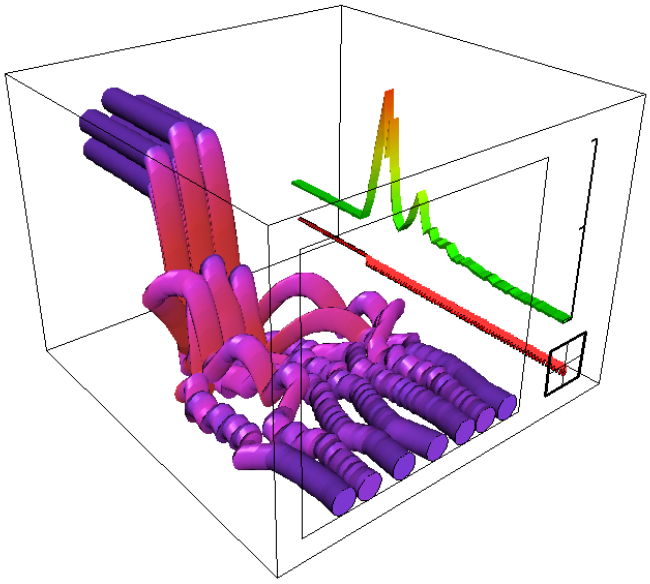

Figure 6: Logged visualization of bouncing balls.

In figure 6, the balls were grouped together and at rest (a field force of zero). At some point in time, the user invoked a particular field force down the y-axis (a gravity). As a result, the balls accelerated and bounced at the bottom and/or with each other. Due to the damping factors and the field force, the balls slowly come to rest at the bottom.

From the displayed log a good impression is obtained of the acceleration of the balls and the increase of kinetic energy after the activation of the field force. Also, the different trajectories followed by the balls can easily be traced.

In figure 7 the use of the clipping plane to browse through the history of the simulation is illustrated. After pausing the simulation, the plane is positioned just after the point of time where the first collisions occurred. The user could restore the positions of the balls as they occurred at this point of time and continue the simulation, for instance with a different field force.

# 5 Conclusion

Logging is an important feature in computational steering. We have extended the graphics tool of the computational steering environment developed at CWI with logging functionality. Since this tool is limited to 2D visualizations, we could use the third dimension as the time-axis and display the logged variables similar to their representation in the current state. More in general, we showed how a 2D user interface can be augmented with logging by using the third dimension for the display of previous states of the interface. We have presented two examples which show the advantages of logging in computational steering: it provides the user with a better insight in relations between variables (an *enhancement* function), it dismisses the user of constantly having to watch the simulation progressing (a *monitoring* function), and it enables the user to jump back to a certain point of time (an *undo* function). However, because this method is only suited for 2D visualizations, the issue on how such a log should be displayed in case of 3D visualizations is still open.

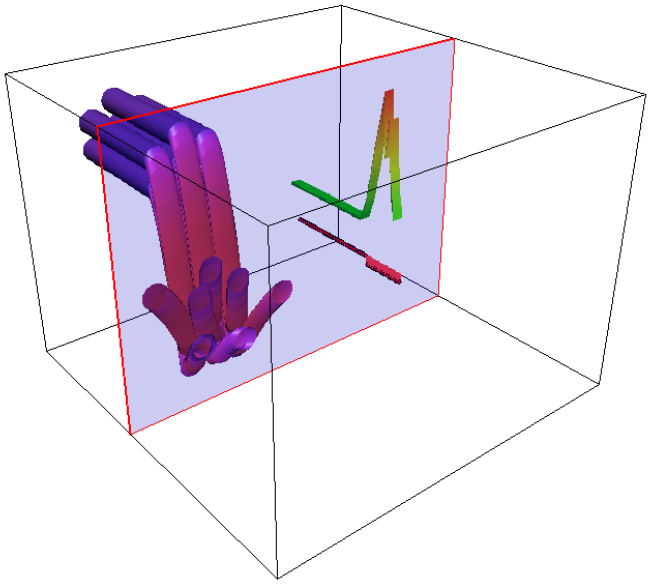

Figure 7: Example of the use of the clipping plane.

## Acknowledgements

The authors would like to thank Robert van Liere (Center for Mathematics and Computer Science, CWI) for his support during the work described in this paper.

## References

[BB93] K. Brodlie and L. Brankin. GRASPARC - a problem solving environment integrating computation and visualization. In G.M. Nielson and D. Bergeron, editors, *Proceedings of the Visualization '93 Conference*, pages 102–109, 1993.

[FvDFH90] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, second edition, 1990.

[MH90] S. Moini and J. Hallquist. Use of concurrent computation and visualization for computation steering in engineering research. In E.J. Farrell, editor, *Extracting Meaning from Complex Data: Processing, Display, Interaction. Proceedings SPIE, 1259*, pages 272–278. The International Society fo Optical Engineering, 1990.

[MKDY90] R. Marshall, J. Kempf, S. Dyer, and C.-C. Yen. Visualization methods and simulation steering for a 3D turbulence model of Lake Erie. *Computer Graphics*, 24(2):89–97, 1990.

[vWvL94] 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.