Bringing Computational Steering to the User

Jarke J. van Wijk
Netherlands Energy Research Foundation ECN
P.O. Box 1, 1755 ZG Petten, The Netherlands
vanwijk@ecn.nl

Robert van Liere, Jurriaan D. Mulder
Centre for Mathematics and Computer Science CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
robertl@cwi.nl, mullie@cwi.nl

Presented at Dagstuhl Seminar on Scientific Visualization
9-13 June, 1997

INTRODUCTION

Visual supercomputing has become a hot topic. Both simulation and visualization have boosted in recent years. Many methods, techniques and tools are available nowadays for the simulation and visualization of complex phenomena. Current computing hardware enables researchers and engineers to perform complex simulations within seconds. The results are visualized with high end graphics workstations, which generate colorful pictures in real-time.

Computational steering is located at the cross-point of these developments. With computational steering the user is continuously provided with visual feedback about the state of his simulation, and can change parameters on the fly. Designers can vary parameters to optimize their product, users can detect errors in the input early, researchers can do qualitative sensitivity analyses easily. Of course, standard methods can also be used: Just perform simulations and display the results afterwards. But we and others [2] believe that computational steering accelerates the process of gaining insight in complex simulations with orders of magnitude, and thereby introduces a new paradigm.

If computational steering is such a great idea, why isn't it routinely used yet? Why are new values for parameters still entered via text files in obscure formats or even by changing the code itself? Why are simulations done until 99.99 % accuracy is achieved, only to observe after the results are converted and displayed with a visualization package that a crucial parameter has a wrong value? The answer is simply that the implementation of computational steering is hard. It requires knowledge of the simulation, visualization, user interfacing, and data communication. An expert in a field such as CFD, molecular dynamics, or mechanical engineering typically will have to cooperate with experts on user interfacing and visualization – which in many cases will be prohibitively expensive. Even if this is possible, it will typically take some months to design and implement an interactive graphics interface. Chances are high that the researcher will have shifted his interest in the meantime. If not, the use of the new interface will almost certainly give rise to new questions as a result of the insight that the researcher obtains. Different parameters need to be controlled, other results must be displayed, so an extended period of close cooperation is required. Consequently, computational steering is currently mainly applied for stable, routinely used applications, where the large effort needed for its implementation is justified.

This is regrettable. Computational steering is especially useful in early development stages of the simulation: when uncertainties exist if the problem is modeled detailed enough; when it is not yet known if the right numerical methods were chosen; when nasty bugs still may be hidden in the code that ruin the results; when many parameters have still to be tuned.

State of the art toolkits for visualization (such as VTK [3]) and user interface development (such as Tcl/Tk) simplify the implementation of computational steering. The developer can assemble components instead of writing everything from scratch. Also, modern visualization packages, i.e. application builders such as AVS and IRIS-Explorer, provide a partial answer. Here the user can define a visualization application via the data-flow paradigm: an application is built up from a collection of modules which inputs and outputs are connected. The simulation can be integrated as a module. This approach has been used for instance by Marshall [2] and in the CUMULVS software infrastructure [4]. They are very positive about the gain in insight and efficiency achieved by computational steering. These approaches offer much flexibility, but their application still seems to require considerable expertise from the developer.

THE COMPUTATIONAL STEERING ENVIRONMENT

Our research aims at the development of general purpose methods, tools, and techniques that enable researchers to implement and use computational steering easily and effectively. In other words, we want to bring computational steering to the user. We target at several audiences. The researcher that de-

fines the model must be enabled to gain insight in the behavior of his model and the numerical methods used. The developer of the simulation code must be enabled to implement computational steering easily. The end-user must be enabled to vary parameters easily and observe the results immediately. In many cases these three roles: researcher, developer, and end-user will be played by the same person. In the remainder of the paper we will refer to him simply as the researcher: an expert in some field, who develops models and code for a class of problems, and who uses this code to study particular problems.

Our research has lead to the Computational Steering Environment (CSE). Basically, the CSE provides a layer between the researcher and a simulation (figure 1). The researcher enters new values via the keyboard or the mouse and views animations of the results. The simulation receives new parameter values and sends updates of calculated results. The main task of the CSE is to communicate and translate the data into comprehensible images, and, vice versa, graphics input into data.

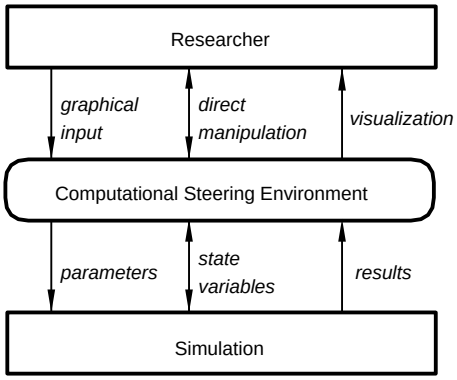

Figure 1: The CSE: a layer between researcher and simulation

We first describe the architecture of the CSE. Flexibility, ease of use, and modularity were the guiding concepts here. Second, the connection of simulations with the CSE is discussed. A simple interface was defined, such that via a few calls data can be exchanged between a simulation and the CSE. Third, we describe how graphical interfaces to simulations can be defined. The concept of Parametrized Graphics Objects enables researchers to easily define both input widgets as well as visualizations. Finally, we present a number of applications. We show that computational steering is an attractive concept, applicable to many different types of applications.

Architecture

Data plays a central role in computational steering. Data is entered, visualized, and exchanged between processes. At least two processes are active: a simulation and a graphical interface. However, more processes could be involved as well: the simulation can consist of multiple processes itself, and additional general purpose tools for data manipulation would come in handy. We have therefore chosen for an architecture with a central *Data Manager* acting as a blackboard, surrounded by *satellites* that produce, process and visualize data (figure 2).

The Data Manager maintains a database of variables. For each variable it stores a name, a type (real, integer, double, string) and value. Variables can be scalar or arrays, in which case the number and size of the dimensions is also stored. Array sizes can change dynamically during the lifetime of the variable. Satellites can create, open, and close variables, and read and write their values. Furthermore, the Data Manager acts as an event notification manager. Satellites can subscribe to events that represent state changes in the Data Manager. Whenever such a state change occurs, the satellite will receive an event from the Data Manager. This mechanism can be used for instance by satellites that monitor changes in the values of variables.

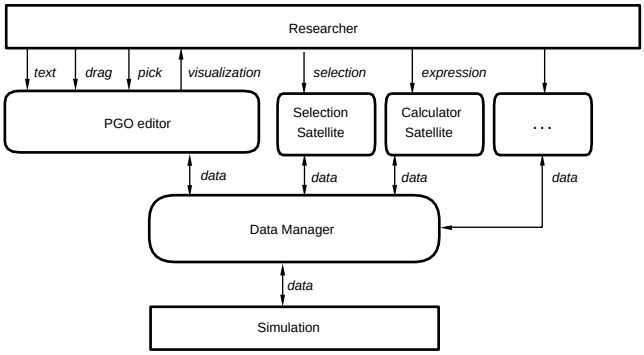

Figure 2: Architecture CSE

This architecture implies that the satellites do not need to know about each others existence. Their only communication with each other is via the variables in the Data Manager. However, connections between satellites can easily be defined by using the same names for variables. Pipelines of satellites (with or without loops) as well as many other configurations can be defined via this mechanism. The number and kind of satellites used during the development or even during a run can be changed dynamically.

This architecture does not provide centralized control: Each satellite decides on its own when to read, process, and write data. However, synchronization can be realized if each satellite has input and output *trigger variables*. When the value of the input trigger changes, a satellite reads input, performs a calculation, and writes output. Finally, the output trigger is written to trigger other satellites [8].

A large collection of general purpose satellites has been developed for standard data processing tasks. For example, data can be logged, sliced, transformed, and calculated. With the constraint satellite constraints can be defined on the values of variables [7]. These satellites produce new derived variables, which can be monitored to gain additional insight. These satellites have a uniform interface, based on a tray of cards, such that their occasional use is straightforward.

The most important satellite is the PGO editor: a general purpose graphics tool for input and visualization of data, which is described later. First we discuss how to connect simulations to the Data Manager.

Integration of simulations

Communication of a satellite with the Data Manager is done via a small application programmers interface. The abstractions used are similar to standard Unix input and output, with variables instead of files. Satellites open variables via a name, and use the returned handle to read, sample, write, and inspect variables. The functionality of this low level interface 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 use the CSE.

What are the requirements ? By far the most important is that the required changes to the simulation code are absolutely minimal. The researcher will not accept to rewrite his (FORTRAN) application according to an Object Oriented approach or to change 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 our solution, the Data I/O library, with an example:

```
simulation(void)
{
   float s;     /* Control parameter */
   float t;     /* Simulation time   */
   float x[64]; /* Results           */
   int n;       /* # Results         */
   int i;       /* Counter           */
   int go_on;   /* Ready or not ?    */

/* Initialize data */

   a     = 0.5;
   t     = 0.0;
   go_on = TRUE;
   n     = 64;

   for (i = 0; i < n; i = i+1)
      x[i] = 0.0;

/* Open Data Manager, declare variables */

   dioOpen("machine.inst.cntry");
   dioConnectFloat("s", &s, READ);
   dioConnectInt("go_on", &go_on, READ);
   dioConnectFloatArray("x", x,
                        1, &n, UPDATE);
   dioConnectFloat("t", &t, WRITE);

/* Main simulation loop */

   while (go_on)
   {
      t = t + 1.0;
      calculate_values(t, s, n, x);
      dioUpdate();
   }
   dioClose();
}
```

The structure of this example is typical for a continuous simulation. First, variables are initialized, next a main loop is entered where time is incremented and new values are calculated. The required changes are limited to opening and closing a connection with the Data Manager, connection of the variables, and a single call to exchange data. The place where to put these calls is easy to locate: 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 sampled. Next for all variables to be written and all unchanged variables to be updated the new values are written to the Data Manager. With these few calls the user is enabled to steer parameters (`s`) of the simulation, to stop the simulation (`go_on`), to monitor its progress (`t, x`) or even to change state variables (`x`). Other satellites can use the `t` as the input trigger: the variable last connected is the variable last written.

This type of simulation continues as fast as possible, without waiting for external events. At each iteration all data is read or written. 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 or any set changes. Hence, a more efficient use of resources can be realized with a small additional effort.

Parametrized Graphics Objects

We now know how to exchange data between simulation and the Data Manager. How about the graphical user interface ? Many tools for defining user interfaces, offering widgets such as text fields, sliders, buttons, etc. exist. Also, many visualization packages, offering graphs, slices, iso-surfaces are available on the market today. One solution is to connect such tools to the Data Manager. As an example, we have developed a satellite to connect IRIS-Explorer to the Data Manager.

However, tight integration of input and output is critical in computational steering. The end-user must be offered a single display, which contains controls for input as well as visualizations of output. For this no off-the-shelf solutions are available, especially if we want to tailor the interface according to concepts used in the application.

As an example, even two scalar parameters can be visualized in many different ways (figure 3). They can be presented via standard user interface widgets such as text fields and sliders, or in a business graphics style. If the parameters denote a position, an interval, angles, or have some other meaning, customized presentations improve the ease of use and insight achieved. And finally, the researcher himself must be enabled to define all this easily.

How to meet these hard demands ? First, the greatest common divisor of user interface widgets and visualization methods

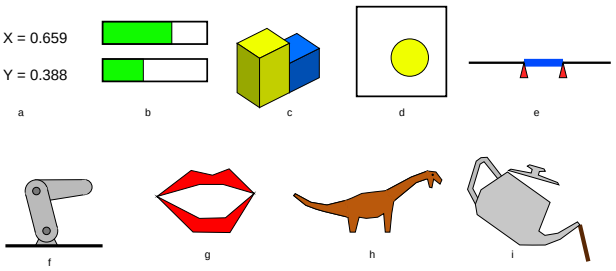

Figure 3: Nine views on two scalar variables

is simply graphics. Buttons, sliders, graphs all boil down to collections of graphics objects. Second, input and output of data can be realized by parametrizing the properties of these graphics objects to data. Hence, our general purpose user interface tool is based on Parametrized Graphics Objects (PGOs).

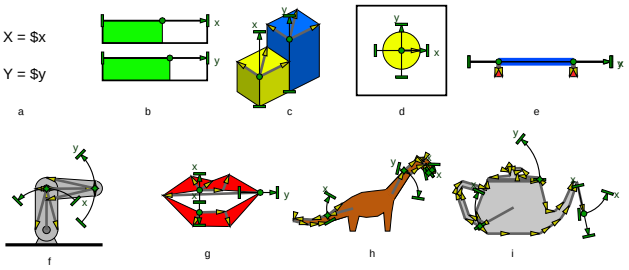

Figure 4: Two scalar variables, edit mode

The PGO editor is an interactive graphics, MacDraw-like, editing tool for the creation and use of computational steering interfaces. With the PGO editor the user creates a visual specification of his interface, just by drawing it. As an example, figure 4 shows how the interface of figure 3 is defined. Both a 3D version [5] and a 2D version have been developed. For tutorial purposes we explain its use via the 2D version. The PGO editor has two modes: specification and application, or shorter, *edit* and *run*. In edit mode, the researcher creates and edits graphics objects, and relates their properties to variables (fig. 5). Graphics objects, such as polylines, rectangles, and circles are defined via points. To these points Degrees of Freedom (DOFs) can be attached. Cartesian as well as polar DOFs are provided. Each DOF defines a range of allowed positions for a point, and has an associated variable. The linear mapping of these geometric ranges to a range of the values of the variable is specified separately. Non-geometric attributes, such as the hue and line width, can also be parametrized. Numerical values of variables can be specified in text objects.

In run-mode, a two-way communication is established between the researcher and his simulation. Data is retrieved from the Data Manager and visualized. Points are moved along the DOFS according to the values of the associated variables, and thereby control the shape, position, size, color and other properties of the graphics objects. Graphics objects

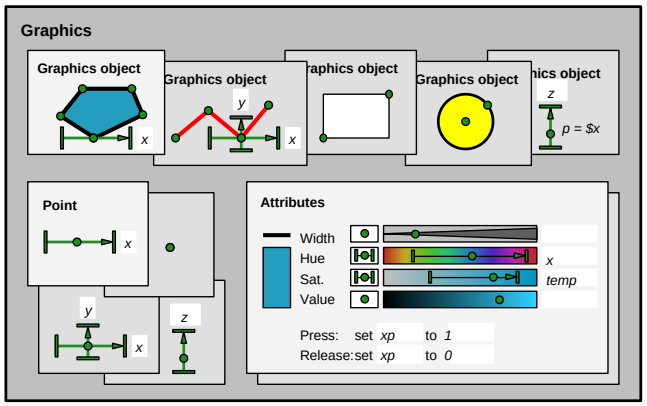

Figure 5: Parametrized Graphics Objects

can be dragged and text can be entered, which is translated into changes of the values of variables in the Data Manager. Note that direct manipulation of objects is supported automatically, which is usually tedious to implement. Graphics objects can be moved by the simulation, but this can be overruled by dragging the object. Picking of graphics objects, for instance needed for the definition of buttons, is also supported. For each object the researcher can specify that a variable has to be given a certain value when picked.

The major part of the scientific data to be visualized will be stored in (multi-dimensional) arrays. Manual specification of a large number of similar PGOs, with DOFs parametrized to indexed elements of the arrays is highly inefficient. Instead, arrays are supported by considering each PGO as a template. If one or more of its DOFs is parametrized to an array variable, multiple instances of the objects are generated, where each is bound to a single entry in the array. As an example: a histogram can be specified by defining a single rectangle, which horizontal position is parametrized to an array with constantly increasing values and which height is parametrized to the variable to be displayed. Polylines, polygons and rectangles can be expanded in two different ways: as separate objects or the points can be expanded inline within a single object. For instance, a graph can be defined by defining an inline expanded polyline consisting of a single point which horizontal and vertical position is parametrized to two arrays of the same length. Inline expanded rectangles denote meshes. The type of mesh generated (equidistant, rectangular or curvilinear) depends on the dimensions of the variables to which the geometry and attributes are parametrized.

APPLICATIONS

We have used the CSE for a number of applications. Here we present a representative sample, other examples can be found elsewhere [5, 6]. These applications vary strongly. The application domains range from pure mathematics to engineering; the amount of interaction varies from monitoring results to full control over many parameters simultaneously; the visual presentation varies from simple graphs to highly customized displays; the main purpose ranged from the estimation of parameters to the demonstration of the model.

In all cases, steering was implemented within a few days, and

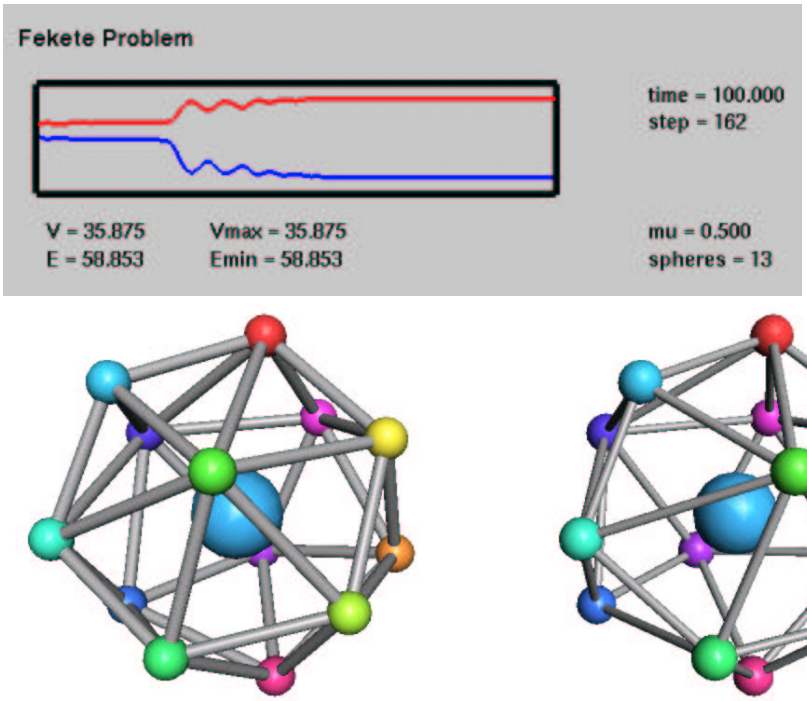

Figure 6: Fekete problem

the results led to much more insight in the simulation. In other words, these examples show that computational steering is a powerful concept, and that our tools provide an environment such that it can be realized easily and effectively.

Fekete problem

Numerical mathematicians at CWI study new methods for the solution of stiff sets of differential-algebraic equations. As an example the Fekete problem is considered. Loosely, the problem is to put a number of points on a sphere such that the distances between these points are maximal. The result can be used for instance as a triangulation of the sphere. More formally, the problem is: Position a set of points $\mathbf{p}_i = [p_{ix}, p_{iy}, p_{iz}], i = 1, \ldots, N$, subject to the constraint $\| \mathbf{p}_i \| = 1$, such that

$$V(\mathbf{p}) := \prod_{i \neq j} \| \mathbf{p}_i - \mathbf{p}_j \|$$

is maximal. This is a global optimization problem for which no closed solution is known [9].

The Fekete problem can be modeled as a set of stiff differential-algebraic equations. Each point is assigned a potential. The differential motion of the points is defined such that the total potential energy P is reduced. An additional adhesion force parameter r is added. It defines how much the points are moved per time step and can be used to make a trade off between fast, but possibly inaccurate results, and slow, precise results. It can be shown that a stable configuration corresponds to a maximum of $V(\mathbf{p})$. The number of points N defines the complexity of the problem; a configuration with N points results in a set of $7N$ coupled differential-algebraic equations. The solution method is based on an implicit Runge-Kutta method.

The steering interface displays a log of the values of V and the potential energy P. One can clearly see that the local minima of P coincide with the maxima of V. Two configurations of the points are shown in the 3D image: the current configuration and the best result achieved so far (figure 6). Here the final result is shown, where these images coincide. Points are depicted as small spheres, and a convex hull is displayed to give a better insight into the three-dimensional structure.

The user can select new values for N and r and watch the progress of the optimization. The animations give the user not only insight into the performance and final results of the solution method for different parameter settings, but especially into the development of the solution. In addition, any point can be dragged to a new position. Small perturbations of points give insight in the stability of configurations.

The number and duration of iteration steps depend strongly on N: for $N = 6$ 105 iterations are needed, which are computed within 7 seconds, $N = 32$ requires 188 iterations which are computed in 90 seconds, results given for an implementation on a SGI Indigo2 workstation with High Impact graphics system. A higher N will require more powerful parallel architectures to solve the governing equations within interactively acceptable time.

Nuclear parameter estimation

The internal structure of the nucleus of atoms is studied intensively. Besides fundamental interests, reliable nuclear data models are essential as input to Monte Carlo codes that are

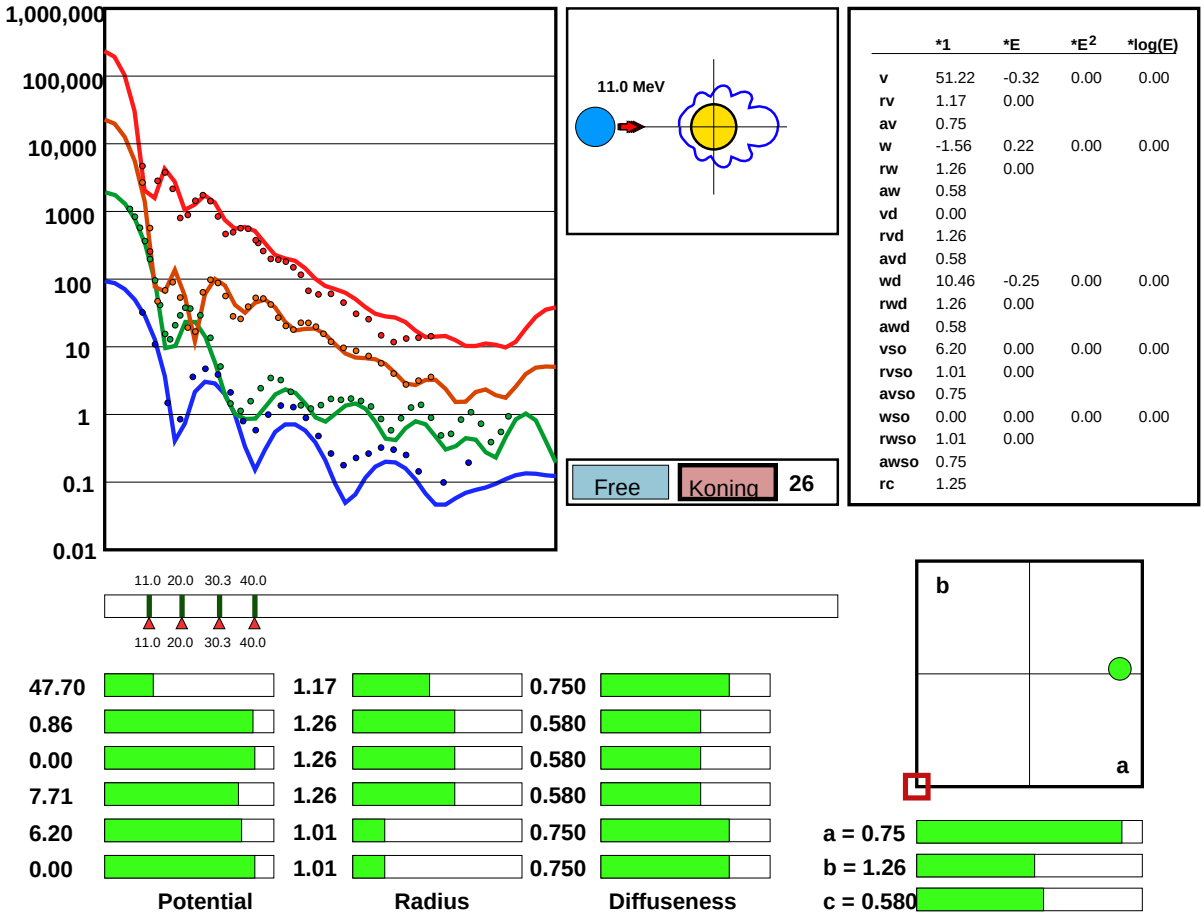

Figure 7: Parameter estimation for optical model

used for the analysis of nuclear applications. The so-called optical model is now widely used as a tool to analyse basic nuclear reactions and the ECIS package [10] (developed by J. Raynal at CEA, Bruyeres-le-Chatel, France) is considered as the evaluation standard. The optical model predicts the probability that a particle (usually a neutron or proton) that enters a nucleus with a certain incident energy will leave the nucleus at a certain angle and outgoing energy. Unfortunately, the optical model has many free parameters that must be determined per type of nucleus. This is a tedious task, requiring many trial-and-error runs to match experimental and model results. The use of automatic optimization is not possible, because a large number of constraints that are hard to formalize have to be satisfied simultaneously. Sound scientific judgement in the parameter estimation loop is essential to reach accurate and reliable results.

The standard way to work with ECIS is to prepare an input-file, do a run, show the results with a plotting package, view the results, prepare a modified input file, etc.. The typical total cycle time is several minutes. We have developed an interactive interface around ECIS to speed up the parameter estimation process (figure 7). The graphs at the left show the calculated results, circles denote measured results. Each graph corresponds to an incident energy. The horizontal axis denotes the angle, the vertical axis the probability (nuclear cross section) on a logarithmic scale. The table at the right shows all parameters. Rows denote various aspects (potentials, radii, etc.), columns denote their dependency on the energy level. The user can change them via the keyboard or via sliders. Two parameters can be changed simultaneously via the widget in the lower right corner. New results are calculated within half a second, hence the user can explore the parameter space interactively. This enables him to gain insight into the sensitivity of the model to certain parameters and to find optimal settings much faster than with standard methods.

Wind turbine model

The interest in wind turbines for the durable production of energy is steadily growing. At the department Renewable Energy of the Netherlands Energy Research Foundation ECN investigations are done on wind turbines by performing structural and aerodynamic tests and by development of design tools. The latter has resulted in the PHATAS computer programme, a general and flexible simulation package for the

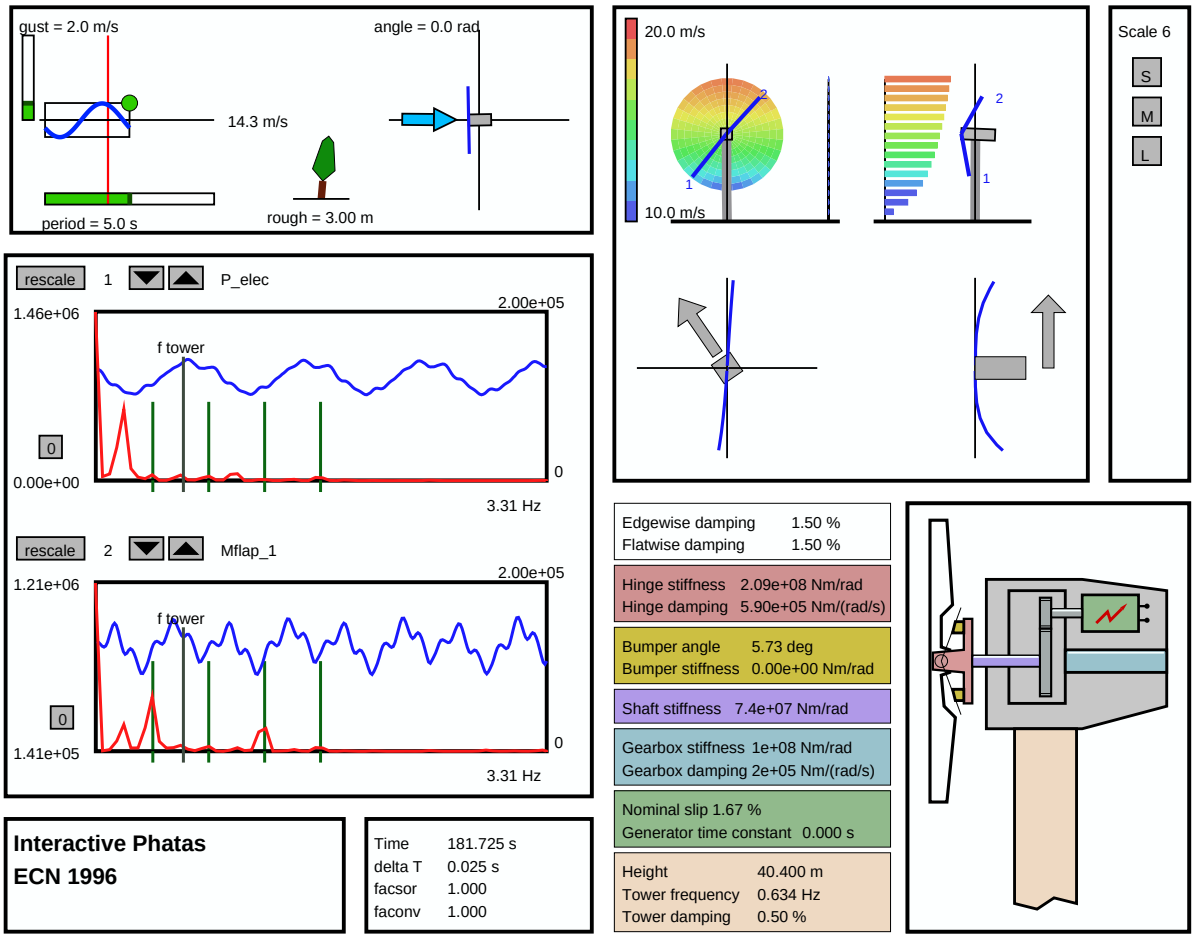

Figure 8: Wind turbine model

evaluation of the mechanical properties of wind turbines [11]. A large number of parameters can be set to define the geometry and material properties of the blades, the drive train and the tower. Wind loading and generator control can be defined in many ways. PHATAS is used for the analysis of problems with existing turbines, and for the development and design of new turbines that optimize economical and safety aspects.

The standard mode of operation of PHATAS is the preparation of an input file and the plotting of results afterwards. However, because the phenomena studied are complex and because many parameters are involved simultaneously, the need arose for a graphical user interface. The result is shown in figure 8. The properties of the wind loading can be set in the box in the upper left corner. Here a deterministic wind model is applied, a sinusoidal variation around an ambient wind-speed, but also a stochastic wind field can be applied. The roughness of the terrain is depicted and can be set via the height of a tree, the direction of the wind by dragging an arrow. The images in the upper right corner show the calculated position and (scaled) deformation of the blades and the tower. Also the current wind field is shown. For the upper images a camera fixed to the world is used, for the lower images a camera fixed to the rotor is used to visualize the deformations better.

Two graph boxes are used in which the user can display a variety of output quantities. Here the electrical power and the flap moment of blade number 1 are shown. Besides the time-series also the Fourier transformed signals are shown (red graphs). These are produced by a separate standard satellite. Furthermore, integral multiples of the rotational frequencies are displayed as green vertical lines, as well as the natural frequency of the tower. These aid the researcher in the correlation of variations in the signal and properties of the turbine.

Furthermore, a schematic sketch of the turbine is shown. The parameters that can be set (mechanical properties, dimensions, etc.) are shown at the left of this sketch.

A simulation is started with an initial set of parameters and continues to run until stopped. Meanwhile the user can change all parameters: to optimize a design; to study sensitivity; to determine critical frequencies; and to study transient effects. On a Alpha workstation a frame rate of 10 frames

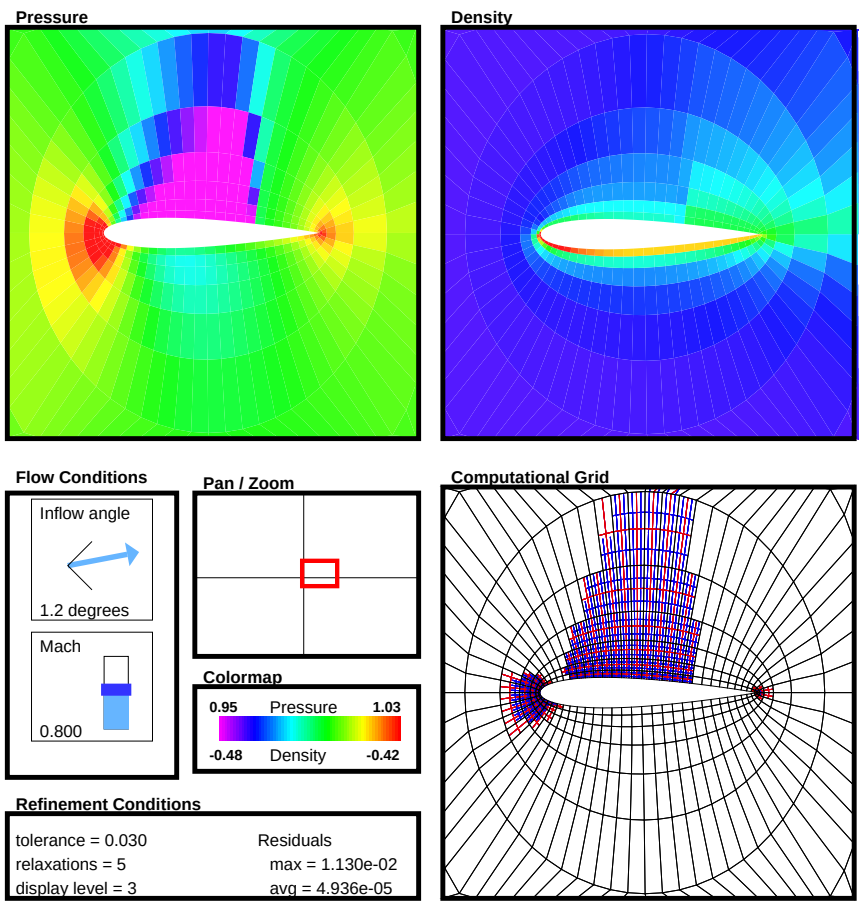

Figure 9: Multi-grid solution of Euler equations

per second is reached, simulation time and wall clock time are almost equal.

Multi-grid solver for the Euler equations of gas dynamics
A classical computational fluid dynamics problem is the analysis of the flow field around an airfoil under various far-field conditions. 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. The efficiency of adaptive multi-grid methods for the solution of systems of partial differential equations is superior to that of other solution methods.

At CWI multi-grid methods are studied intensively, especially for the solution of the compressible Euler equations [12]. Figure 9 shows an interface to the multi-grid flow solver. 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. New parameter values define a completely new problem, so the solver is reinitialized each time a parameter is changed. Intermediate results of the solver are displayed to show the convergence of the process. Furthermore, 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 possibly inaccurate results, and slow, precise results.

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

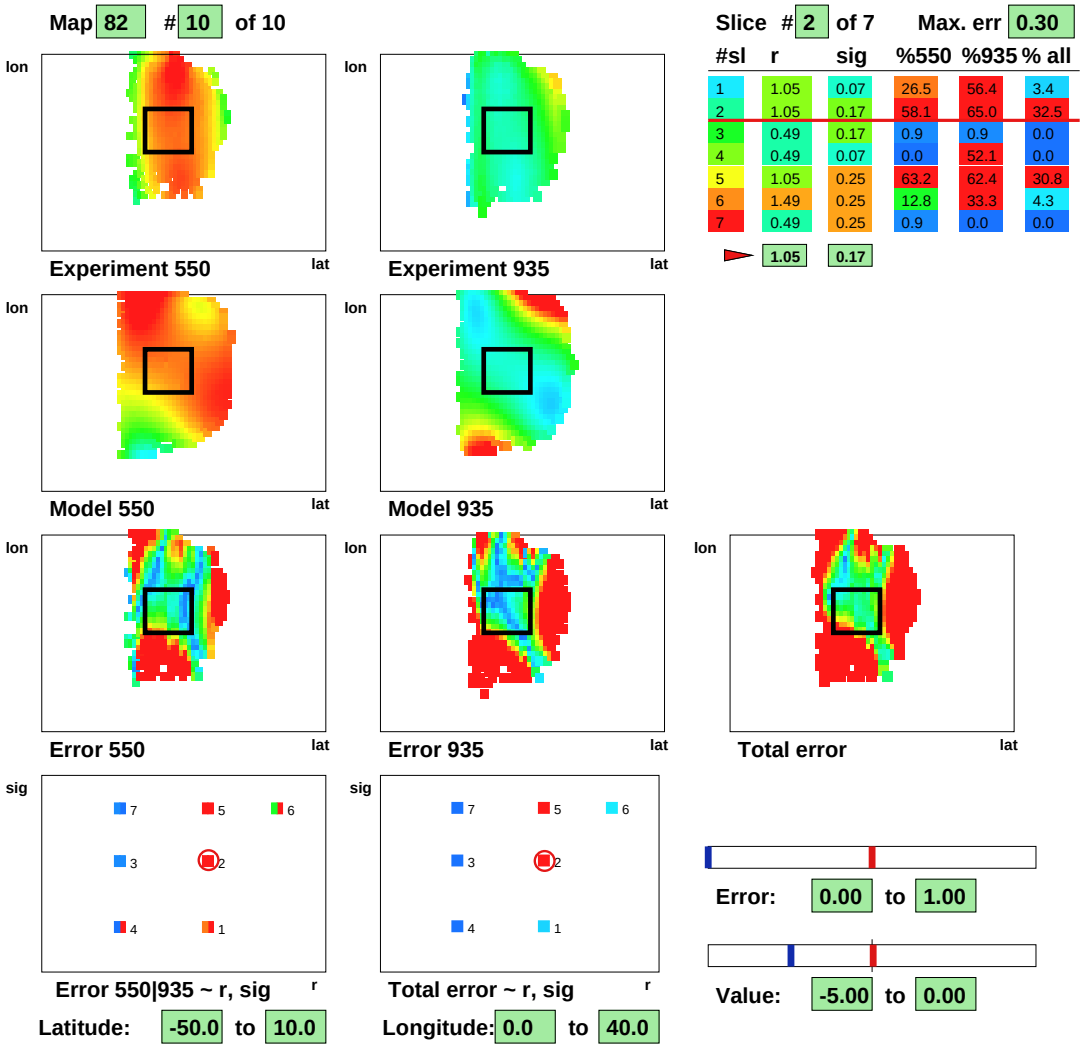

Figure 10: Light scattering of Venus

Light scattering of Venus

The Pioneer Venus Orbiter has encircled Venus for more than a decade, and gathered huge amounts of light scattering data: intensities and degree of polarization for four different wavelengths. Knowledge of the theory of multiple light scattering has allowed for the development of compute intensive programs which calculate the afore mentioned experimental parameters for model atmospheres. These models take into account one or two species of spherical particles with sizes characterized by their effective radius r and their effective standard deviation σ. A human-in-the-loop approach to gain insight and to determine values for these parameters is necessary here, because physical constraints and a priori knowledge of the researcher are hard to formulate explicitly, yet must be satisfied to guarantee realistic results.

A user interface was developed for researchers of the Vrije Universiteit, Amsterdam, which study light scattering intensively [13]. An more extensive description can be found elsewhere [14]. Straightforward computational steering was not applicable here, since a single simulation takes already considerable time on super computers (30 s on a Cray C98/4256). Therefore, a separate satellite was developed which maintains a database of all results calculated so far. The user selects a particular day and two wavelengths, upon which all results for different combinations of r and σ are looked up and summarized in a table and in diagrams (Figure 10). Furthermore, the user can select new values for r and σ, upon which a new simulation is started. The spatial distribution of experimental and simulation results are presented, for two different wavelengths, as well as the difference between them. The user can also select older results for fast comparison. This interface enables researchers to navigate through the parameter space easily and effectively. The reuse of previously computed results, their simultaneous visualization in different ways, and options for easy browsing were important keys to realize this.

DISCUSSION

With these applications we have shown that our solution enables researchers to implement and use computational steer-

ing easily and effectively without much support from user interface experts. An architecture with a central Data Manager gives flexibility and modularity. A minimal subroutine interface is presented to communicate data between Data Manager and simulation. With parametrized graphics objects custom user interfaces, both for input and output, can be defined by the researchers themselves. The CSE thus enables researchers to develop interfaces iteratively, together with the development of their model, simulation, or input. When new parameters have to be controlled or other results must be visualized: it is easy to connect new parameters and to define their visualization.

The development of CSE and its application to a variety of cases has learned us much about the applicability and usefulness of computational steering. And, the more we apply computational steering, the more positive we become. Our experience is that researchers are most often eager and willing to use it, especially if a first feasibility study can be set up within a few hours. Some state initially that computational steering is not applicable for their particular application, because their program takes 10 minutes or more before final results are produced. However, usually an internal loop can be detected in which results are refined, and visualization of these intermediate results often gives insight within a few seconds. Also, the use of control parameters for accuracy improves interactivity: low values give faster update rates.

Computational steering is useful in several areas. The standard application of computational steering is parameter variation such that the researcher achieves insight. We have shown such applications, but others are maybe even more important. In one confidential case the use of CSE was crucial to debug a complex numerical code. Presentation is another area. Technical discussions with colleagues progress much faster if "What if?" questions can be answered immediately. The development and use of mathematical models is for outsiders (i.e. the management) often an obscure activity. Attractive graphical interfaces enable researchers to communicate their models and results on a level that is much closer to that of the audience. Computational steering can provide a transparent window on the real world, in all its complexity.

In our future work we aim at further improvement and expansion of the CSE. First, the resource management can be improved. All data sets are now routed via pipes to the Data Manager. For small data sets this works well, for very large data sets more efficient techniques are required. The conceptual model has proved to be simple and effective for users, hence these optimizations must be hidden for the users. Second, the current CSE provides only support for direct steering. We want to develop additional tools that support the user in his navigation in parameter space. The GRASPARC environment [15] is a good example in this respect. Third, we will explore presentation techniques further, especially for 3D simulations. PGOs provide an excellent means for fast implementation of icons and composite 3D interactors.

ACKNOWLEDGEMENTS

We would like to thank Jacques de Swart (CWI), Koert Lindenburg (ECN), Barry Koren and Eric van der Maarel (CWI), Arjan Koning (ECN), and Hans Spoelder and Willem-Jan Knibbe (VU) for their pleasant cooperation. This work is partially supported by the ICES-HPCN programme, and by the Netherlands Computer Science Research Foundation (SION), with financial support of the Netherlands Organization for Scientific Research (NWO). More information on CSE can be found in http://www.cwi.nl/projects/cse/cse.html.

REFERENCES

1. McCormick, B., T. Defanti, and M. Brown, Visualization in Scientific Computing. *Computer Graphics*, 22(6): p.103–111, 1987.
2. Marshall, R.E., J.L. Kempf, D. Scott Dyer, and C-C Yen, Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie. *1990 Symposium on Interactive 3D Graphics*, *Computer Graphics*, 24(2): p.89–97, 1987.
3. Schroeder, W., K. Martin, and B. Lorenson, *The Visualization Toolkit An Object-Oriented Approach to 3D Graphics*, Prentice Hall, 1996.
4. Kohl, J.A., P.M. Papadopoulos, A Library for Visualization and Steering of Distributed Simulations using PVM and AVS. Presented at the AVS User Conference, 1995.
5. Mulder, J.D. and J.J. van Wijk, 3D Computational Steering with Parametrized Geometric Objects. In: Nielson, G.M. and D. Silver (eds.), *Proceedings IEEE Visualization'95*, CS Press, pp. 304–311, October 1995.
6. Liere, R. van, J.D. Mulder, and J.J. van Wijk Computational Steering. In: Liddel, H., A. Colbrook, B. Hertzberger, and P. Sloot (eds.), High-Performance Computing and Networking, Lecture Notes in Computer Science, Springer-Verlag, vol. 1067, pp. 696–702.
7. Wijk, J.J. van, A Model for Strategy in Constraint Solving. Submitted for publication.
8. Liere, R. van, and J.J. van Wijk, CSE: A Modular Architecture for Computational Steering. In: P. Slavik, and J.J. van Wijk (eds.), *Proceedings of the 7th Eurographics Workshop on Visualization in Scientific Computing*, Prague, April 1996.
9. Pardalos, P.M., An Open Global Optimization Problem on the Unit Sphere. *Journal of Global Optimization*, 6, 1995, p. 213.
10. Raynal, J., *Notes on ECIS95*. CEA Saclay report, 1995.
11. Lindenburg, C. and H. Snel, PHATAS-II: Program for Horizontal Axis wind Turbine Analysis and Simulation version II. In: S.M. Hock (ed.), Wind Energy 1993, Proceedings of the 16th Annual Energy-Sources Technology Conference, Houston, Texas, 1993, SED-Vol.14, ASME, New York.
12. Hemker, P.W., B. Koren, W.M. Lioen, M. Nool, and H.T.M. van der Maarel, Multigrid for Steady Gas Dynamics Problems. In: Deconinck, H. and B. Koren (eds.), *Euler and Navier-Stokes Solvers Using Multi-Dimensional Upwind Schemes and Multigrid Acceleration*, Vieweg, Braunschweig, 1996.

13. Knibbe, W.J.J., J.F. de Haan, J.W. Hovenier, and L.D. Travis, Spatial variations of Venus' cloud properties derived from polarimetry. In Atmospheric Sensing and Modeling, R. Santer (Ed.), SPIE Proceedings, 2311, pp. 47 - 57, 1994.

14. Wijk, J.J., H.J.W. Spoelder, W.J.J. Knibbe, K. Eftekhari Shahroudi, Interactive Exploration and Modeling of Large Data Sets: A Case Study with Venus Light Scattering Data. In Yagel, R., and G.M. Nielson, *Proceedings of IEEE Visualization'96*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

15. Brodlie, K., L. Brankin, G. Banecki, A. Gay, A. Poon, and H. Wright, GRASPARC: A problem solving environment integrating computation and visualization. In Nielson, G.M., and D. Bergeron, *Proceedings of IEEE Visualization'93*, pp. 102-109, IEEE Computer Society Press, Los Alamitos, CA, 1993.