Nonlinear Analysis: Modelling and Control, 2003, Vol. 8, No. 2, 71–82

FDVis: the Interactive Visualization and Steering Environment for the Computational Processes Using the Finite-Difference Method*

A. Kurtinaitis[1], R. Vaicekauskas[2], F. Ivanauskas[3]

[1]Vilnius University, Naugarduko 24, 2600 Vilnius, Lithuania
andrius.kurtinaitis@maf.vu.lt

[2]Vilnius University, Naugarduko 24, 2600 Vilnius, Lithuania
rimantas.vaicekauskas@maf.vu.lt

[3]Vilnius University, Naugarduko 24, 2600 Vilnius, Lithuania
Institute of Mathematics and Informatics, Akademijos 4, 2600 Vilnius, Lithuania
feliksas.ivanauskas@maf.vu.lt

Received: 24.07.2003
Accepted: 30.09.2003

Abstract. In this paper a specialized software environment for visualization and steering of finite-difference computations is presented. The user requirements are identified and the architecture of the environment is summarized. The advantages of such a specialized system over some available universal visualization systems are discussed and conclusions and future research issues are given.

Keywords: scientific visualization, computational steering, finite-difference, numerical simulation.

1 Introduction and problem statement

Every computational simulation produces a set of numbers. If this set is not very large, one can easily comprehend it or compare it with the expected results. However, there are cases, where one cannot compute short characteristics for the modeled phenomena or such short characteristics are not known. Then we have to look through the large amount of numerical data and try to

*This work was supported by Lithuanian State Science and Studies Foundation, project No. C-03048.

intuitively analyze it. The larger the amount of data is, the harder one can understand the meaning of this data. Scientific visualization can make the numerical data visible to the human eye. Scientific visualization is a computational process that transforms the data objects of scientific computations into visible images on a computer display screen [1].

Often it is not enough to see the final result of the computations. The intermediate data can also be of major importance. It could happen, that for certain values of input parameters of the computation, the desired result will be computed earlier than expected. It could also happen, that at the very beginning of the simulation process the intermediate result shows, that further simulation makes no sense. Therefore, it is also important to have a possibility:

- to observe the state of the computations;
- to stop the simulation and to change the values of the input parameters.

These properties of a simulation system are called computational steering. An interactive steerable software system can save much computing resources and time of the scientist. Authors and users of steerable systems also note that the possibility to observe the state of the computations and to interactively change the input parameters allows to develop more intuition about the effect of problem parameters. It also helps to detect program bugs, to develop insight into the operation of the algorithm, or to deepen an understanding of the physics of the problem being studied [2].

Most traditional numerical computation systems use separate unrelated programs for the computation and for its visualization. The visualization of the resulting data in such a system can be a lengthy and tedious task, because it demands much manual work of the scientist. He may be required to import the data into the visualization program. It can also be necessary to convert the data to a format, which the visualization program can handle. The scientist needs to specify how the data should be displayed. The required user actions to visualize the data are almost identical in a series of a similar experiments, therefore it is preferable to automate them.

Universal interactive numerical computation environments build a possible automated solution. Such a system incorporates all the processes of the numerical experimentation: geometric modeling, specification of the input data,

computations and the visualization – everything is running under the control of one program. The advanced instances of universal computation environments are *SCIRun* [2] and *CUMULVS* [3]. Some scientific software packages also provide a means to create an environment for visualization of computations. Those are: *MATLAB* [4], *SCILAB* [5] and *Octave* [6]. The third possibility to create such an environment is to use scientific visualization frameworks. *The Visualization Toolkit* [7] and *VisAD* [8] are widely used ones. A detailed survey of the existing computational steering and visualization environments can be found in [9]–[11].

Every group of a universal solutions we mentioned above has their own drawbacks:

- They are difficult to adjust and apply for a particular problem. The adjusting requires often some programming. This is especially true for scientific software packages.
- Universal systems are very complex and therefore they require much time to learn before one can use them.
- They are not portable.
- They provide a limited user interface (scientific software packages).
- They are expensive and are not accessible to many scientists.
- The computing performance is not always good, especially when using interpreted code in scientific software packages.

Our goal is to create an interactive visualization and steering environment for finite-difference simulations which overcomes the trade-offs of the available universal systems listed above. One way to do this is to restrict a set of the problems which could be solved with the help of the environment. The more specialized environment is easier to learn. It can also provide a much better user interface because it should not take into account the requirements of every possible numerical simulation problem, which is the case with a universal environment.

The presented software system *FDVis* is an efficient interactive visualization and steering environment specialized for computations based on the finite-difference method. In this paper we outline the major steps of creating simple scientific computation environment and share the experience we earned and

the lessons we learned. First we identify common properties of the finite-difference computations, which allow to simplify the environment. Then we discuss usual requirements of the scientist and usual experimentation scenarios which together form the requirements for the software system being created. We demonstrate the use of the environment for solution of real problems and define a class of problems for which it is useful.

2 Common properties of the finite-difference algorithms

The finite-difference method is a method for solving differential equations. A numerical problem which is expressed as a differential equation, always consists of:

- the differential equation, which describes the modeled process;
- the physical area, where the problem has to be solved;
- the initial and boundary conditions.

In the beginning, the numerical characteristics of the modelled phenomenon at the boundaries and the relations of these characteristics inside the area are known. These relations are expressed by the means of (a system of) differential equations. When solving the equations using finite-difference method, the differential operators of the equations are approximated by the finite-differences on certain area points – on the grid. Using boundary conditions and difference equations we build a system of algebraic equations (often linear) to find values of the unknown grid points. Most applications of the finite-difference method compute new grid points in subsequent layers according some dimension of the area. Using this property of layered computation, we can state the following assumptions for the visualization and steering system:

- the results of the finite-difference computation are split in layers;
- the computing program works as a filter transforming the known values of the previous layer to the new layer;
- the layers are computed sequentially one by another;
- every layer is an intermediate result of the computation, it represents some step of the computation along some axis.

We also noticed another property which holds for most of the finite-difference computations. The finite-difference method is used for problems with relatively simple geometry of the physical area and unsophisticated initial and boundary conditions (usually given by some analytical formulas). Although theoretically this method can be used with problems with physical areas and boundary conditions of any complexity, in such cases other methods are mostly used, for instance, finite elements. When creating the visualization and steering system *FDVis*, we followed the following principles:

- We do not need a geometric modeling subsystem to specify the geometry of the area. It is enough to give some scalar parameters: length, width, step count and so on.
- To specify boundary and initial conditions of the problem it is also enough to give some scalar parameters, which define the boundary values or the constants in formulas if the boundary conditions are specified as some analytical formulas.

3 User requirements

When performing numerical experimentation, the following scenario is usually used by the scientist: first the input parameters for the computation are specified, then the necessary computations are performed and finally the numerical results are visualized and analyzed. To achieve maximum usability of the computation environment, all the steps mentioned above should be accessible using a consistent user interface. Following the three steps of the experimentation, we specify further requirements for the user interface of the system being created:

3.1 Input parameter specification

- any finite number of scalar input parameters can be specified;
- every input parameter should have its unique name;
- the parameters can be of various types; at least integer, real and string valued parameters must be supported;

- the input parameters for the entire series of experiments can be specified by varying the value of some selected parameters;
- input parameters can be specified as expressions of other parameters; for instance, by taking the value of time step parameter equal to the step along some space axes or by specifying physical value of some parameter but using the normalized value of it.

3.2 Computations

- the computations can be observed by inspecting the intermediate numerical results;
- the computations can be aborted without loosing the already computed intermediate results;
- the series of computations can be started using the specified input parameters for all the series.

3.3 Visualization

- Many characteristics of the numerical results can be visualized at the same time. For instance, by modeling the generation of the ultrashort laser pulses it is important to see the duration of the pulse and its energy distribution on some plane of the crystal.
- During the computation one can select which of the many available characteristics of the numerical results should be visualized.
- During the computation one can choose one of the many visualization techniques for the selected characteristics. For instance, two-dimensional array of the intensity values of the laser pulse can be visualized as a three-dimensional surface, as a set of two-dimensional iso-lines or as a color map.
- The visualization should be possible not only during the computations, but also after them, using saved numerical result data. The system should allow to select the experiment out of the series and the numerical characteristic which should be visualized.
- The results of the visualization should be usable for the further processing – for creating presentations, publications and so on.

The integrated visualization and steering software system should provide a comfortable user interface to meet the requirements listed above.

4 System architecture

When creating the interactive visualization and steering environment *FDVis*, we followed the properties, which are common to all finite-difference computations: (1) the results are computed in layers and (2) the geometry of the problem area and the boundary conditions can be specified relatively simply. The previous section defines the functional requirements for the software system. We have also some non-functional requirements. One of them is the overall system efficiency. The possibility to interactively visualize and steer computations should not create any significant slowdown of the computations itself. The system should be flexible enough and easily adaptable to the different problems. To separate the parts of the system which can be exchanged by adapting to the new numerical problem, we decided to split the system into individual components (Fig. 1).

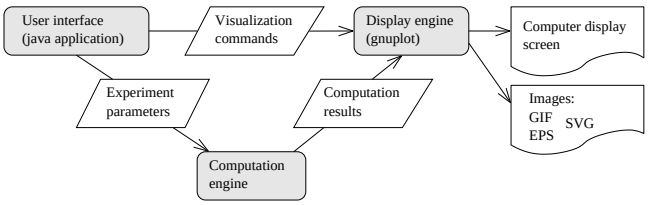

Fig. 1. *FDVis* system architecture overview.

The first module which can be different for another numerical problem is a computation engine. It is a program that performs numerical computations which simulate the problem being solved. Usually, the computation engine is created by the expert of the problem domain. We created the computation engine for *FDVis* which simulates the generation of ultrashort laser pulses [12]. Its executable *shgsolve* reads the initial laser pulse characteristics as input parameters and computes the intensity and other characteristics of the laser pulse in a number of crystal layers. The essential requirement for the computation engine is the efficiency. Therefore the computations are usu-

ally implemented using the compiled programming language like C, C++ or FORTRAN. There are cases where we may want to use an old computation engine in the visualization system. Such a computation engine cannot have any information about the visualization and steering system attached to it. We use a computation engine as a separate executable module, which reads the file with the input parameters and in the process of computation, generates the files with the numerical result data of the layers along some dimension of the numerical problem area.

Another component of the system *FDVis*, which performs a clearly defined function and is potentially separable, is the display engine. It is a module, which transforms the numerical data into the images on the computer display screen. This transformation is often numerically intensive, so it is also important that the module is implemented in an efficient way. The efficient visualization of the numerical data is a very well researched area. There are many software packages for data visualization – both free and commercial. So we decided not to reinvent the wheel and use one of those packages – the *gnuplot*. The following properties of *gnuplot* have determined the choice:

- It is functional. It can display both two-dimensional and three-dimensional data in many different ways.
- It is fast (written entirely in C).
- It is portable (written in portable C).
- It is flexible and easily used as a utility program.
- It is free. It costs nothing and is free to modify to fit our needs.

When needed the *gnuplot* can be easily replaced with another visualization package which meets some specific needs, because it is also a separate executable module and is interfaced only through its standard input stream.

The third component of the *FDVis* system, which binds all the system together, is the user interface module. It knows the interfaces of other two components and does the actual steering. It observes the progress of the computation, waits for the new numerical results and passes them to the display engine for visualization. It also provides graphical user interface which allows to work with the numerical experiments in a convenient way:

- to define the series of the numerical experiments;
- to specify the input parameters for the experiments;
- to start and to stop the experiments;
- to observe the progress of the computations;
- to select the characteristics for visualization.

Because the usability and portability of the user interface is more important than its speed, we implemented the user interface module in the interpretative high level language *Java*. At the moment it is probably the only solution, providing comfortable user interface capabilities on many computer platforms.

The three components of the system are used as the separate executable modules. The communication between the modules is performed using the file system and the standard input/output streams. All the exchange of the data is performed using standard textual ASCII format. This way of integration provides many advantages. Some of them are: simplicity, good support of different operating environments and easier verification and debugging of the system.

The main development work was performed on a Linux platform using mostly free UNIX development tools and utilities. According to the recomendations for the application development for the UNIX operating environment, described in [13], the programs should perform only one function and exchange the information in a clearly defined and simple way. Then these programs are easy to combine and form a qualitative new system. These recomendations are also particularly useful when creating a system for numerical visualization and steering.

5 Application example

FDVis system has been successfully used when modeling ultra-short laser pulses using second-harmonic generation in nonlinear environments [14]. The equation system we solved is presented below:

$$\frac{\partial A_l}{\partial z} + a_l \frac{\partial A_l}{\partial t} + i b_l \frac{\partial^2 A_l}{\partial t^2} + \frac{i c_l}{r} \frac{\partial}{\partial r}\left(r \frac{\partial A_l}{\partial r} \right) = i d_l \varphi_l + e_l A_l. \qquad (1)$$

Here $A_l(r,t,z)$ are complex valued functions, l=1, 2, 3; a_l, b_l, c_l, d_l and e_l are real constants; φ_l are nonlinear functions, depending on A_1, A_2 and A_3:

$$\varphi_1 = A_2^* A_3 e^{-i\kappa z}, \quad \varphi_2 = A_1^* A_3 e^{-i\kappa z}, \quad \varphi_3 = A_1 A_2 e^{i\kappa z}. \tag{2}$$

System (1) was solved in the area $Q = [0,R] \times [0,Z] \times [0,T] \subset \mathbb{R} \times \mathbb{R} \times \mathbb{R}$.

The picture below shows the common view of the computational workbench when running one of the experiments: A typical computational work-

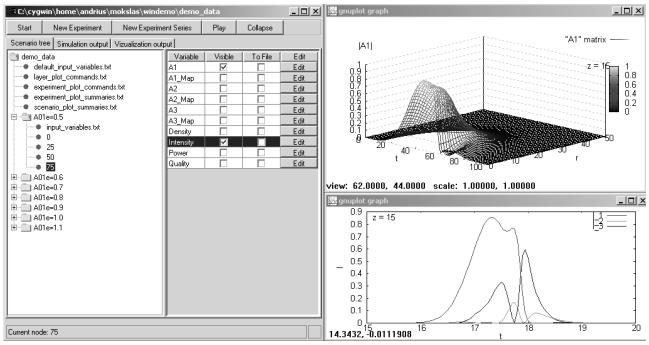

Fig. 2. A typical computational workbench when using *FDVis* system.

bench when running one of the experiments is depicted in Fig. 2. The GUI component on the left side of the picture allows to observe the status of the experiment called "$A01e = 0.5$". Currently the layer "75" is ready, laser pulse characteristics (visual views) "A1" and "Intensity" are selected. The right side of the screen shot contains rendered plots of the selected characteristics: "A1" as a surface, "Intensity" as three curves on one plot area.

6 Conclusions

In this paper we described the creation of *FDVis*. The result of the work is a fully functional system with a comfortable user interface. It allows the scientist with the minimal or even without programming knowledge to perform the usual numerical simulation tasks: to define a series of experiments, to run and steer the computations and finally to analyze the results and use the resulting data for the publications.

Although the system *FDVis* was develop specially for the computations based on the finite-difference method, it is also useful for other computations, which use a similar data model and provide the data in layers. The methods in question include finite-elements and some others as well.

Our experience has shown that an intuitive user interface is very important for numerical simulation work. It allows to perform all the needed simulation and analysis tasks by the expert of the problem area without the need of a programmer. It also helps to see the result the algorithm's input parameter changes more quickly and to develop some intuition about it.

The goal of the work described by this paper was to create a simple yet useful tool for scientific visualization and steering of the finite-difference simulations. The specialized system we created has the basic functionality of visualization and steering system. However, there is a number of ways how we can further enhance the system. Some of the desirable features are already addressed by other, universal visualization and steering systems.

It could be possible and useful to capture more semantical information about the underlying numerical data. The visualization could then be done in a more automated way. The system could perform some analysis of result data structures and automatically reject the inappropriate ways to visualize it. The user would be supplied with a choice of suitable visualization alternatives. This problem is deeply researched in the works of Hibbard [1, 8].

There would also very nice to have a possibility to continue the stopped computations from some position with possibly new parameters. This feature is already implemented in some integrated computation environments like *SCIRun*.

Also, no possibility to perform visualization and steering of the distributed computations was considered. Taking into account, that we are using portable and loosely-coupled system components in *FDVis*, the extension in that direction is also thinkable.

References

1. Hibbard W. *Visualizing Scientific Computations: A System based on Lattice-Structured Data and Display Models,* Univ. of Wisc. Comp. Sci. Dept., 1995

2. Johnson C., Parker S. “The SCIRun Parallel scientific Computing Problem Solving Environment”, *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999

3. Geist G.A., Kohl J.A. Papadopoulos P.M. “CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications”, *International Journal of High Performance Computing Applications*, **11**(3), p. 224–236, 1997

4. *The MathWorks*, Inc. Getting started with MATLAB, 2002, http://www.mathworks.com/access/helpdesk/help/techdoc/learn_matlab/learn_matlab.shtml

5. Gomez C. *Engineering and Scientific Computing with Scilab*, Birkhauser, Boston, 1999

6. Eaton W. *GNU Octave Manual*, Network Theory Ltd., 2002

7. Schroeder W., Martin K., Lorensen B. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*, Prentice-Hall Inc., Upper Saddle River, NJ, 1998

8. Hibbard W., Rueden C., Emmerson S., Rink T., Glowacki D., Whittaker T., Fulker D., Anderson J. “Java distributed objects for numerical visualization in VisAD”, *Communications of the ACM*, **45**(4ve), p. 160–170, 2002

9. Mulder J.D., van Wijk J.J., van Liere R. “A survey of computational steering environments”, *Future Generation Computer Systems*, **15**(1), p. 119–129, 1999

10. Parker S.G., Johnson C.R., Beazley D. “Computational Steering Software Systems and Strategies”, *IEEE Computational Science & Engineering*, **4**(4), p. 50–59, 1997

11. Gu W., Vetter J., Schwan K. “An Annotated Bibliography of Interactive Program Steering”, *ACM SIG-PLAN Notices*, **29**(9), p. 140–148, 1994

12. Kurtinaitis A., Dementjev A., Ivanauskas A. “Modeling of pulse propagation factor changes in type II second-harmonic generation”, *Nonlinear Analysis: Modeling and Control*, **6**(2), p. 51–69, 2001

13. Ganzarc M. *The Unix Philosophy*, Digital Press, 1995

14. Dement’ev A., Ivanauskas F., Kurtinaitis A. “Modeling of compression dynamics and change of pulse quality during the type II second harmonic generation”, *Proceedings of the XV-th Byelorussian-Lithuanian seminar on Lasers and optical nonlinearity*, p. 74–82, 2002