

GRASPARC - A Problem Solving Environment Integrating Computation and Visualization

Ken Brodlie
Andrew Poon
Helen Wright

School of Computer Studies
University of Leeds
Leeds, UK

Lesley Brankin
Greg Banecki
Alan Gay

NAG Ltd
Oxford
UK

Abstract

Visualization has proved an effective tool in the understanding of large data sets in computational science and engineering. There is growing interest today in the development of problem solving environments which integrate both visualization and the computational process which generates the data.

The GRASPARC project has looked at some of the issues involved in creating such an environment. An architecture is proposed in which tools for computation and visualization can be embedded in a framework which assists in the management of the problem solving process. This framework has an integral data management facility which allows an audit trail of the experiments to be recorded. This design therefore allows not only steering but also backtracking and more complicated problem solving strategies.

A number of demonstrator case studies have been implemented.

1 Introduction

Visualization has become a key tool in computational science and engineering. The 'firehoses of data' predicted by the NSF Report (McCormick et al [10]) can now be directed at one of a number of powerful visualization systems, and the scientist can gain an understanding of their data that would not previously have been possible.

Generally, however, the computation process which generates the data is seen as quite separate from the visualization process which views it. The next challenge is to study how to provide an environment in which the two are combined, and where the scientist

thinks in terms of the overall task of solving a problem, not simply of viewing the results, and in which a distributed computing environment can be exploited. This paper describes the GRASPARC project, which has looked at some of the issues involved in creating such a problem solving environment.

2 Visualization and Problem Solving Environments - Some Limitations

We begin by looking at the use of existing visualization systems as problem solving environments.

Fundamental work by Upson et al [14] and Haber and McNabb [7] has established a model for data visualization that has underpinned many current software systems. This model decomposes a visualization task into a set of modules; the modules belong to one of four classes: data sources; filters - to refine the data; mappers - to construct an abstract geometrical representation of the data; and renderers - to generate an image.

This modular decomposition is the basis of a number of popular visualization systems: AVS [14], IRIS Explorer [5], IBM Data Explorer [8], apE [4] and Khoros [13]. These all provide a visual programming front end, in which the user can select appropriate modules for an application, connect them together in a network and trigger execution by activating a data source. Data passes through the network of modules, from filters to mappers to renderers, finally generating an image - see Figure 1 - hence the term *dataflow system*. Each module is typically controlled by a set of parameters - for example, the threshold value in an isosurface mapping module; user interface widgets associated with each module allow the user to modify

these parameter values, and thus adapt interactively the flow of data through the network until a required view is obtained.

In early versions, the data source modules simply read data from a file and so there was no possibility of a direct link to the computation process which created the data. It was a natural extension however to allow application code, for example Computational Fluid Dynamics (CFD) software, to be incorporated as a 'data source' module. This gives rise to the term *application builder*, where the visualization system houses both computation and visualization - a complete application. A user interface can be created for the embedded computational code; parameters of the application can be changed interactively allowing a simple form of computational steering (see Marshall et al [9], for example) - as shown in Figure 1.

Thus such application builders can act as simple problem solving environments. They have necessarily however grown out of visualization systems, rather than be designed from the outset as problem-solving environments. Thus while there is support for carrying out *one* simulation, or a simple sequence of simulations with different parameter settings, there is typically little or no support for the complex search process that a scientist typically undertakes in modelling some phenomenon. For example, there is generally no data management facility to record the results of one simulation for later comparison; it is difficult to back-track from one point in a simulation, and restart from an earlier point with some change in parameters.

3 The GRASPARC Project

The GRASPARC project has attempted to work at the outset from a problem solving point of view. We are interested in constructing an environment in which the computational scientist or engineer can 'plug in' the tools they need, and in which there is an integral data management facility to provide an efficient audit trail of the experiments carried out.

The work has involved a number of strands which are discussed in more detail in the later sections of this paper:

Reference Model: An overall view of the mathematical modelling process has been developed, in order to gain a high-level understanding of the processes involved in problem solving.

Modelling the Search for a Solution: The next level of refinement is to understand the search

process undertaken by the scientist in solving a problem. A model has been built which sees the process as a *tree structure*: the root of the tree represents the start of a series of experiments; a *branch* is created at some point where a new experiment with different parameters is begun; along any branch are *snapshots* at which results are recorded and from which new branches can be created. The final tree is the complete audit trail of the set of experiments. The user interface to our system then becomes an interface to this model.

Architecture - Components and Framework:

From an understanding of the problem-solving process, it becomes possible to identify the components needed to provide a good working environment: computation and visualization components provide the tools, but these need to be embedded in a framework that will allow them to work in harmony, and communicate in a way which is transparent to the scientist. Most importantly, there is a need for a data manager to hold the tree data structure which underlies the search process.

High-quality numerical tools and excellent visualization software already exist, so it is the goal of GRASPARC to provide the framework in which they can be integrated.

The animation of sequences of images is seen as a fundamental requirement for visualization, and thus we are investigating as part of the project, special-purpose animation hardware.

Throughout the project we have kept in touch with a range of potential users: computational chemists, meteorologists, fluid dynamicists, and mechanical engineers. These contacts have enabled us to understand the requirements of problem solving environments, and they have provided a source of demonstrator studies which have been used as test-bed for our ideas.

4 A Reference Model for Problem Solving

The work began by developing an overall view of the mathematical modelling process, through a series of interviews with our user contacts.

The model is described in terms of a three-dimensional structure involving two parallel planes:

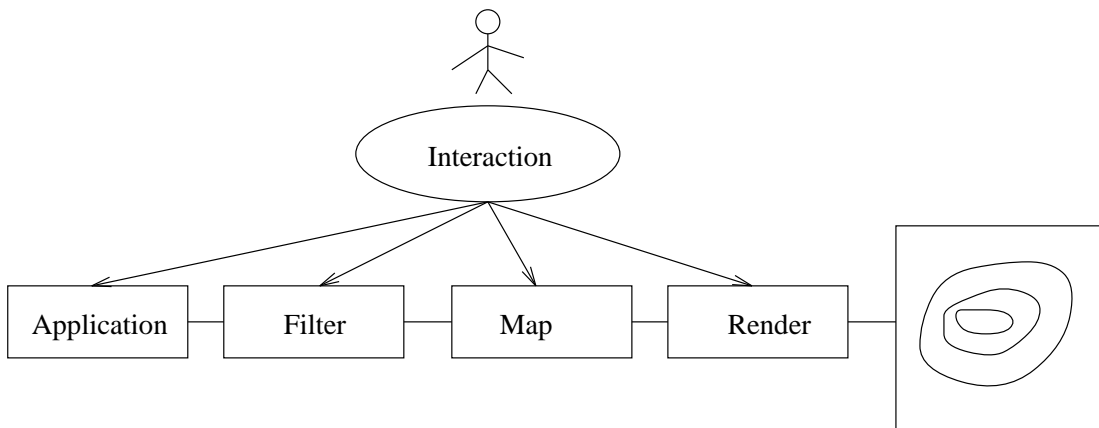


Figure 1: Steering in the Dataflow Environment.

an Investigator's Plane which holds the cognitive activities associated with problem-solving (such as assimilation of results); and a Simulation Plane which holds the states through which a problem passes en route to solution (see Figure 2).

The Simulation Plane is subdivided into layers of increasing specificity: the upper layer is the Model Layer in which the problem is expressed in some idealised form; in the middle layer, termed the Continuum Layer, the problem is expressed as a functional specification, say a set of differential equations; the lower layer, the Discrete Layer, contains a discretised formulation of the equations to be solved numerically. Lower layers are invoked as necessary when a solution is not directly available within the present layer.

Between the planes lie tools for presentation and interaction. Tools for presentation provide the user with a representation (which may or may not be visual) of what is occurring in the simulation plane, whilst tools for interaction provide the means for influencing the simulation.

A complete description of the GRASPARC reference model is given by Wright et al [15]. GRASPARC concentrates on the lower layer of the model - namely, the search for a solution of the numerical simulation.

The next level of refinement of the GRASPARC model is to understand the search process undertaken by the scientist within the Discrete Layer. But first we describe a simple case study which will help explain this search process.

5 Case Study - Kinetics of Thermally Activated Processes

This is a problem in computational physics, described by Cartling [3]. A particle is moving in a potential and connected to a heat bath. The potential has two minima corresponding to the states of the system between which transitions take place - the physicist is interested in how the particle behaves over time, for different strengths of coupling to the heat bath.

The motion is described by the Fokker-Planck equation:

$$\frac{\partial P}{\partial t} = -v \frac{\partial P}{\partial x} + \frac{1}{m} \frac{dU}{dx} \frac{\partial P}{\partial v} + \beta \left(\frac{\partial}{\partial v} (vP) + \frac{\kappa T}{m} \frac{\partial^2 P}{\partial v^2} \right)$$

where:

- $P = P(x, v; t)$ is a probability density function, giving the probability of the particle having position, x , and velocity, v , at time t ;
- $U(x)$ is the supplied potential (bistable);
- β is a damping factor.

Initially at time $t = 0$, the probability distribution is a sharp peak centred on $x = -2, v = 0$; that is, the particle has high probability of being stationary at one minimum of the potential. The other minimum is at $x = +2, v = 0$. As t increases, so the distribution changes, and will vary depending on the value of β . Figure 3 shows the contours of P as time progresses - the plume indicates passage of the particle over the potential barrier separating the two minima.

In the Discrete Layer of our reference model, the problem is formulated as the numerical solution of the Fokker-Planck equation, which in our case is

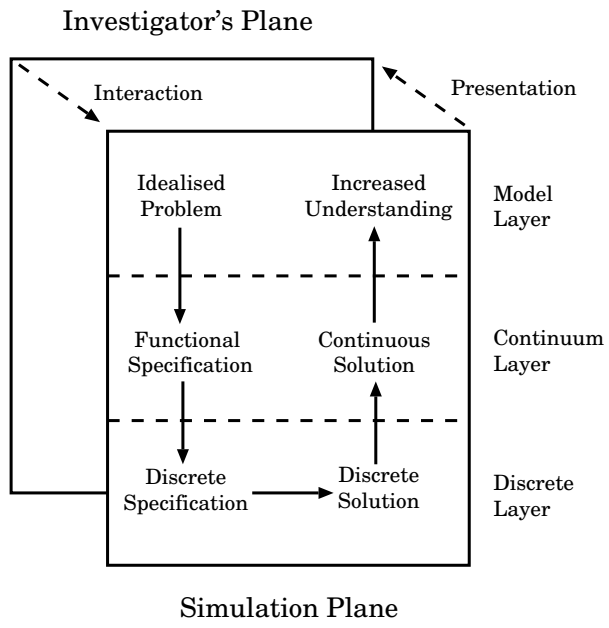


Figure 2: The GRASPARC Reference Model.

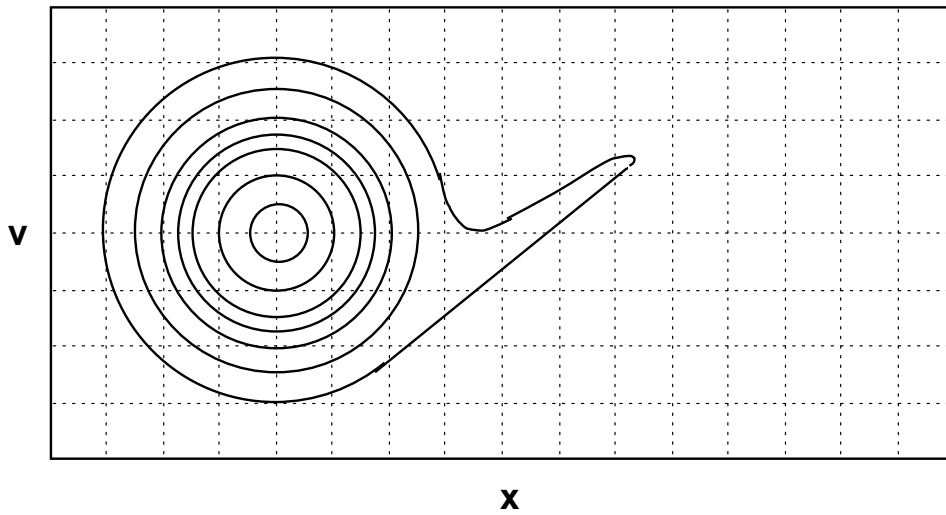


Figure 3: Fokker-Planck Solution : $P(x, v)$ at time $t > 0$.

achieved using the SPRINT differential equation software (Berzins et al [2]). The PDE's are discretised in the two space dimensions, generating a set of ODEs which are solved by SPRINT.

The computational scientist will want to control a number of aspects in solving this problem:

- β , the strength of coupling to heat bath
- mesh size for spatial discretisation
- frequency of output of results

There is interest in comparing results for different values of β ; and in halting a calculation in order to backtrack some time steps to repeat the calculation with a different mesh size to gain greater accuracy - both operations which could not easily be achieved simply by embedding the numerical software in a visualization system.

6 Modelling the Search for a Solution

The fundamental concept in GRASPARC is the model of the search process by which a scientist investigates a problem in the Discrete Layer. We see the computational process as an ordered set of events in time. These events might be points at which an intermediate solution can be output - or at which the computation can be interrupted and some parameter changed before restarting. This can be modelled as a tree structure, which we have termed a *History Tree* (see Figure 4).

To understand the History Tree, consider the solution of the Fokker-Planck case study described earlier. The scientist will begin solution of the equations with a certain mesh size, and with requests for the solution to be recorded every five seconds, say. These outputs are the events mentioned above. If the solution is visualized as it is calculated, the scientist may spot some feature of interest - or notice its absence. This is a signal to halt the computation, and explore this more closely. Rather than go back to the start and repeat the entire calculation at higher accuracy, the scientist would prefer to reload the solution from a few time-steps previous, and resume from there with finer mesh size and more frequent output of results.

This is the model of working we aim to support in GRASPARC. The overall set of events for a computational experiment is described as a *history tree*. The tree will have a set of associated parameters, which

specify the experiment - here the Fokker-Planck equations, the potential within which the particle is moving, and so on. Each *branch* of the tree corresponds to a particular decision by the scientist - for example, to set the time increment for output. Thus each branch will also have an associated set of parameters, specifying the particular options chosen by the scientist at that decision point. Finally each branch consists of a sequence of *snapshots* - the individual events along a branch, essentially the iteration steps towards a solution. Data stored at a snapshot may be at different levels of abstraction: either the raw data, or a geometrical representation, or an image representation. Alternatively an event may be recorded as having taken place, but with no actual data stored.

7 GRASPARC System Architecture - Framework and Components

The Reference Model has helped identify the scope of the GRASPARC project as providing an environment in which problem solving in the Discrete Layer can be effectively managed. The History Tree has given us a model of the process of searching for a solution. The next step has been to define a basic system architecture, comprising a framework into which different components or tools can be inserted.

The GRASPARC architecture is shown in Figure 5. The framework provides a support environment for one or more numerical and graphical applications. It consists of the following elements:

GRASPARC Management System (GMS):

This is the principal component of the design. Its basic data structure is the History Tree, recording the process by which the scientist is iterating to a solution of the problem. Its function is to accept user commands and formulate these in terms of a basic set of operations on the tree - for example, 'create a new branch' or 'visualize a snapshot'. Thus the GMS is essentially a model of the problem solving process as seen by GRASPARC. The output from the model is a set of instructions to the different components in the system - for example, creating a new branch will generate an instruction to a numeric component to restart from a given point with a new set of parameters.

The GMS has been designed to allow concurrent processing of the basic set of operations - thus it is possible to 'explore' multiple branches simultaneously.

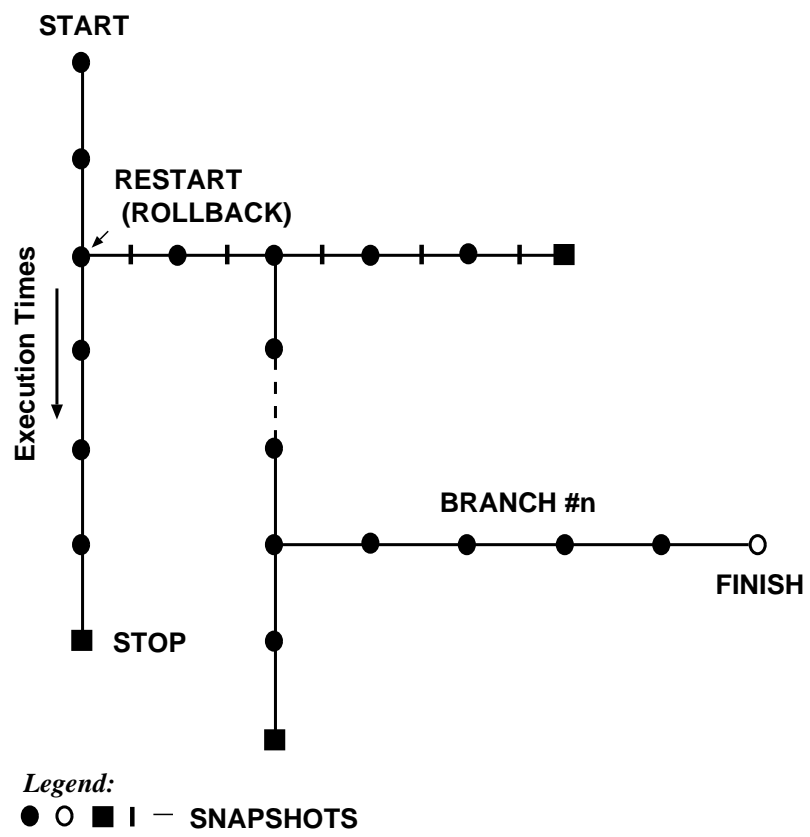


Figure 4: The GRASPARC History Tree.

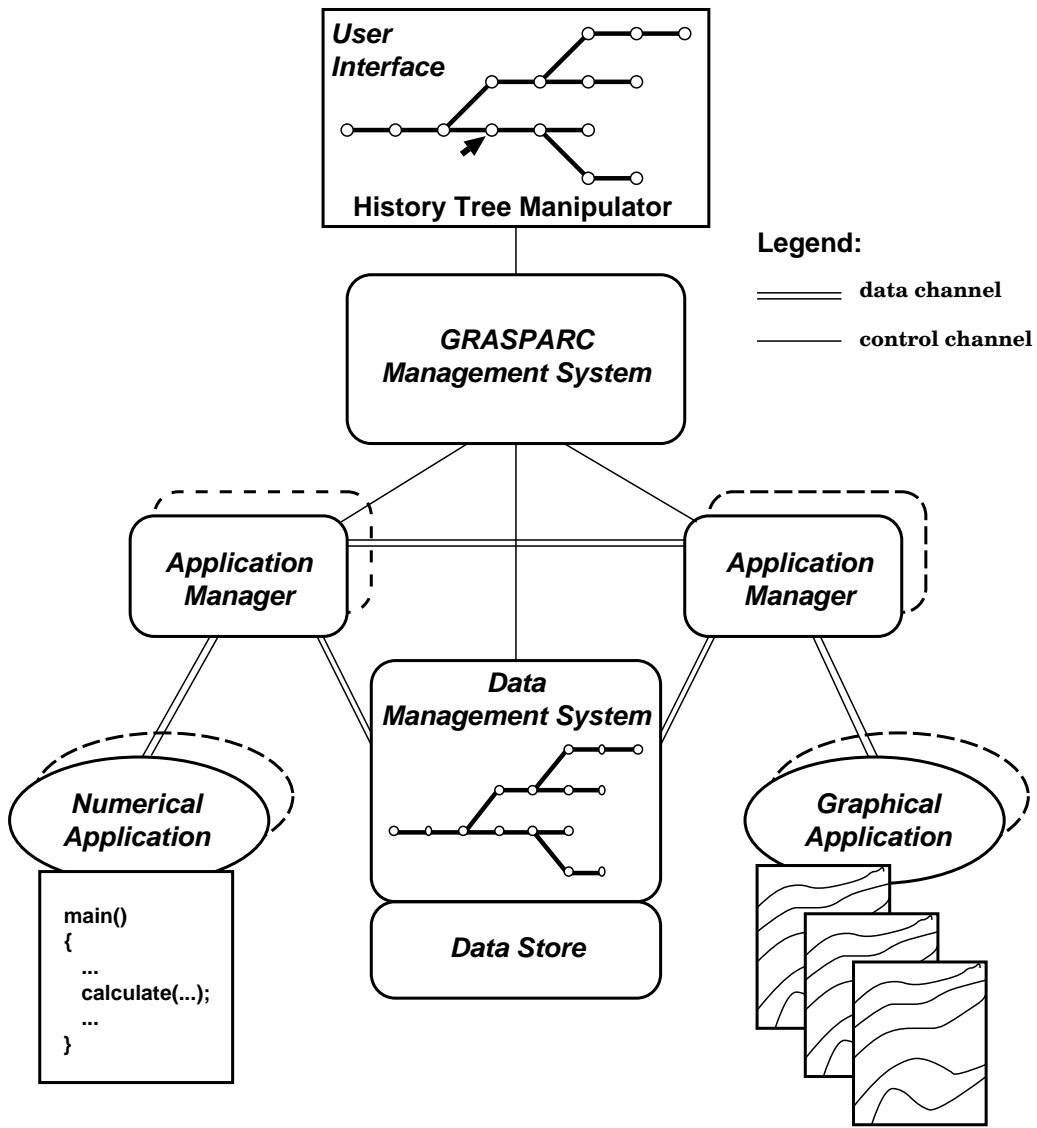


Figure 5: The GRASPARC Architecture.

GRASPARC User Interface (UI): The GRASPARC user interface is likewise modelled on the History Tree - indeed it is essentially a representation of the current state of the History Tree as known by the GMS. The scientist thus thinks in terms of a search process with snapshots, branches and trees. A 'tree viewer' displays the structure and allows the scientist to select a snapshot as a new branch point, or to select a sequence of snapshots for visualization.

GRASPARC Data Management System (DMS):

GRASPARC includes its own dedicated data management system, which can be logically viewed as a subsidiary function of the GRASPARC Management System.

It is designed as a layered model. The lowest layer, called the *physical data layer*, contains a set of primitive data types (such as reals, integers and strings) and has been implemented in terms of the Hierarchical Data Format (HDF) [11].

The middle layer, called the *composite data layer*, contains a set of composite types derived from the primitive types: such as different mesh structures, images, etc. The definition of this layer has been influenced by the requirements of the case studies examined during the project, and also by the datatypes from existing visualization systems, particularly IBM Data Explorer (see Haber et al [6]).

The top layer, called the *data model layer*, is essentially the History Tree data structure, in which the composite types are organised in a tree hierarchy.

A set of logical operations for the storage, extraction and removal of data from the tree have been developed. Note that the GRASPARC Management System simply holds a map of the structure of the History Tree - it is the Data Management System which holds the actual data.

Thus we see that the History Tree is the fundamental concept around which the GMS, the UI and the DMS are all organised.

GRASPARC Application Managers (AM):

Each numerical or graphical component is attached to the framework by means of an application manager, whose function is to translate data and commands between the native format of the application and a common GRASPARC format. It is the responsibility of the problem solver to

develop the AM, but GRASPARC provides tools, guidance and interface definitions.

An important aspect is the separate data and command highways. Command highways link the AMs to the central GMS; data highways link AMs of components directly - so computational results can be passed on a 'fast track' to the graphical component, or the DMS. This separation means that the command highways do not get choked with data, and instructions can be passed rapidly to components.

A component or its AM can include a user interface to control the setting of parameters for that component - for example, initial conditions for solution by a numerics component.

Thus a GRASPARC problem solver will be built by identifying suitable numerical and graphical software for the problem at hand - this is seen as the responsibility of the problem owner. These tools are brought into the GRASPARC environment, by connecting them to the framework as described.

The architecture allows the components to be distributed across different processors - so that the computation components for example can run on a high performance computing node and the graphics components on one or more workstations.

Special-purpose animation hardware is being developed by the other partners in the project, Quintek Ltd. This is treated as a graphics component, and connected via an AM as described above. Images are retrieved from the DMS (by selecting from a history tree display of available images), stored in a cache and replayed as requested.

8 Developing GRASPARC Demonstrators

As with any research project, the ideas described above have been conceived, formulated and matured as the project has progressed. In order to test the feasibility of the concepts, a number of demonstrators have been constructed during the project.

The first prototype was built to solve the Fokker-Planck case study mentioned earlier. The numerical component was the SPRINT software; the graphical components were contouring routines, based on the FARB-E software (Preusser [12]), and surface view routines written locally - both based on an implementation of PHIGS [1], the ISO standard for 3D graphics.

The system was implemented on a network of Silicon Graphics 4D/240 and Personal IRIS workstations.

Figure 6 shows an example from a problem solving session. The scientist is looking at two different concurrent graphical representations of a snapshot. Notice that they have backtracked to an earlier snapshot to create a new branch.

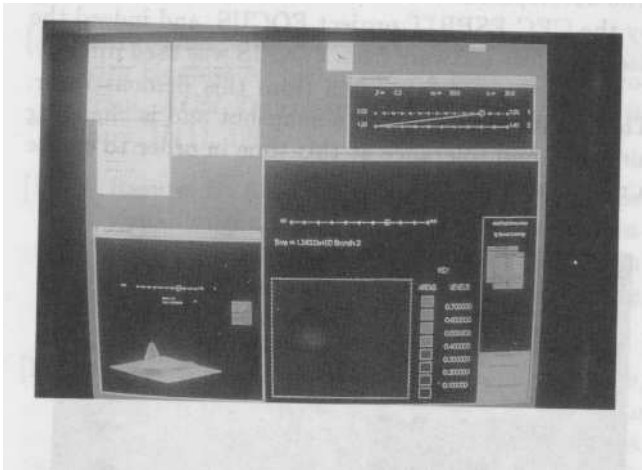


Figure 6: Solving the Fokker-Planck Problem.

The next demonstrator looked at a problem in aeronautics. A wedge is put in the path of a jet of fast-moving air; the flow cannot continue smoothly over the wedge and a shock front develops because the air is necessarily compressed into a smaller space in order to be able to pass the obstruction - the shock front represents a line at which all solution variables change rapidly in space - see Figure 7.

The scientist may wish to experiment with altering the angle of the wedge, or the velocity of the air. During the solution, the scientist may wish to halt the computation, return to an earlier solution time and resume calculation with a finer tolerance, so as to better pinpoint the position of the front.

This demonstrator was built again using SPRINT-based numerical software, but this time using the dataflow visualization system, AVS [14], as the graphical component. The GRASPARC Application Manager which interfaces AVS to the framework, passes a script to AVS so as to automatically generate a suitable network based on the data to be displayed - thus shielding the scientist from the workings of AVS itself. The development of this demonstrator was influenced by the CEC ESPRIT project FOCUS, and indeed the user interface software from FOCUS was used directly.

Figure 8 shows a session from this demonstrator. The scientist has selected a snapshot and is changing

air speed and tolerance at this time in order to create a new branch.

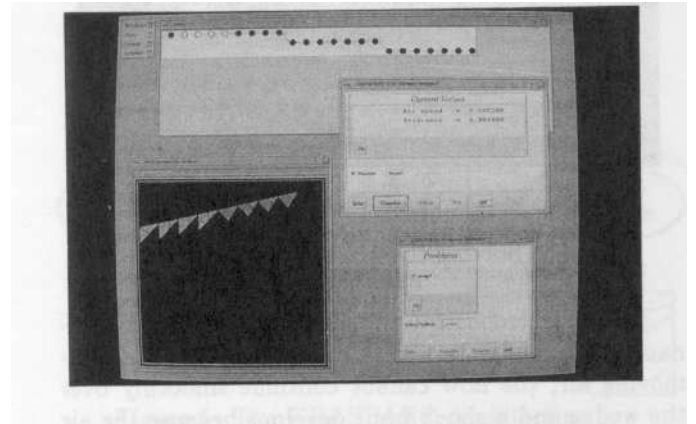


Figure 8: Solving the Wedge Problem.

Two further demonstrators are under development - from the fields of chemical reaction simulation and planetary motion - involving different numerical and graphical components. In these cases, as in the demonstrators just described, the scientist will be able to exploit the GRASPARC architecture in exploring different solution options in a well managed way.

9 Conclusions and Future Work

GRASPARC has defined a model of a problem solving environment, in which numerical and graphical components can be integrated under a common framework. This framework is based on the History Tree concept, which reflects the search process used by a scientist in reaching an optimal solution to a simulation. There is an integral data management facility which allows an audit trail to be recorded.

The study has led us along a number of avenues which are still being explored. For example, the close coupling of computation and visualization raises important issues for direct manipulation of a simulation, and for the accuracy of visual representation.

Taking direct manipulation first, consider again the Fokker-Planck problem and suppose we overlay the computational mesh on the solution as it is visualized. Rather than globally request a mesh refinement, one can point at particular areas of the mesh where refinement is needed. This requires the input of spatial data back to the simulation - a facility rarely provided in current systems.

As far as accuracy is concerned, we are looking at the ways in which 'functional' information held in the

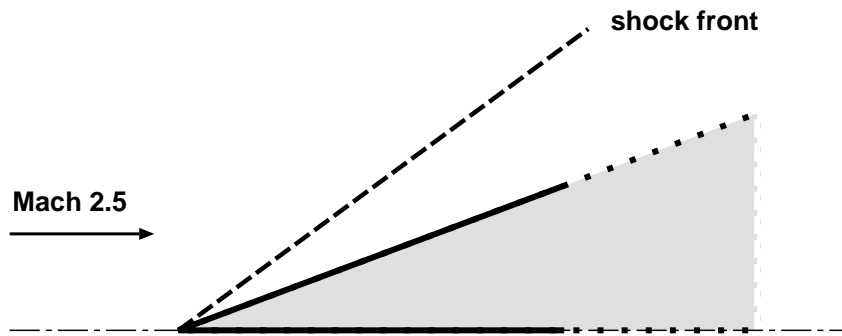


Figure 7: Flow over wedge creating shock front.

numerical component can be transferred to the graphics component. In conventional systems, the interface is at the level of data - with the graphics system independently recreating the interpolant with consequent loss of accuracy.

We are still at the early stages of research into problem solving environments. We hope GRASPARC may influence the subsequent development of commercial systems - whether they grow out of existing visualization systems or are developed *ab initio*.

The next step is to seek an environment in which a host of tools can be employed, not just numerical and graphical, but also computer algebra, computational geometry and documentation systems - to provide a 'virtual laboratory' for the computational scientist.

Acknowledgements

GRASPARC (GRaphical environment for Supporting PARallel Computing) is a three year (17 person year) collaborative project funded jointly by the UK Department of Trade and Industry and the Science and Engineering Research Council. The partners are NAG Ltd, The University of Leeds (School of Computer Studies) and Quintek Ltd.

Many people have contributed to the thinking behind GRASPARC, and to its successful management as a project. In particular, we would like to thank Steve Hague of NAG Ltd, who has been a most supportive and effective project manager throughout; and Peter Dew and Martin Berzins of Leeds University, Richard Brankin of NAG Ltd and Pat Mills of Quintek Ltd, who have given invaluable technical direction to the project.

Others have contributed in many different ways: we would like to thank Victoria Pennington, Justin Ware, Neil Bowers and Gary Stead of Leeds University; Jimmy Brown and Del Cornali of NAG Ltd; and Mark Powell and Phil Copeland, who have worked for

Quintek Ltd during the project. We would also like to thank our DTI Project Officer, Dennis Henn, for his unstinting support.

Finally we must thank the many users who have helped us through discussions and by attendance at workshops, and ensured we retain contact with the real world.

References

- [1] ISO/IEC 9592/1. *Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System - Part 1 - functional description*. ISO/IEC, 1988.
- [2] M. Berzins, P.M. Dew, and R.M. Furzeland. Developing software for time-dependent problems using the method of lines and differential-algebraic integrators. *Applied Numerical Mathematics*, 5:375-397, 1989.
- [3] B. Cartling. Kinetics of activated processes from non-stationary solutions of the fokker-planck equation for a bistable potential. *J. Chem. Phys.*, 87(5):2638-2648, 1987.
- [4] D. S. Dyer. A dataflow toolkit for visualization. *IEEE Computer Graphics and Applications*, 10(4):60-69, 1990.
- [5] G. Edwards. Visualization - the second generation. *Image Processing*, pages 48-53, 1992.
- [6] R.B. Haber, B. Lucas, and N. Collins. A data model for scientific visualization with provisions for regular and irregular grids. In *Visualization '91 Proceedings*, pages 298-305. IEEE Computer Society Press, 1991.

- [7] R.B. Haber and D.A. McNabb. Visualization idioms : A conceptual model for scientific visualization systems. In B. Shriver G.M. Nielson and L.J. Rosenblum, editors, *Visualization in Scientific Computing*, pages 74–93. IEEE, 1990.
- [8] B. Lucas, G.D. Abram, N.S. Collins, D.A. Epstein, D.L. Gresh, and K.P. McAuliffe. An architecture for a scientific visualization system. In A.E. Kaufman and G.M. Nielson, editors, *Visualization 92 Proceedings*, pages 107–114. IEEE Computer Society Press, 1992.
- [9] R. Marshall, J. Kempf, S. Dyer, and C. Yen. Visualization methods and simulation steering for a 3d turbulence model for Lake Erie. *ACM SIGGRAPH Computer Graphics*, 24(2):89–97, 1990.
- [10] B. McCormick, T.A. DeFanti, and M.D. Brown. Visualization in scientific computing. *ACM SIGGRAPH Computer Graphics*, 21(6), 1987.
- [11] NCSA. *HDF Calling Interfaces and Utilities*. National Center for Supercomputer Applications, University of Illinois at Urbana-Champaign, 1989.
- [12] A. Preusser. Algorithm 671 - farb-e-2d: Fill area with bicubics on rectangles - a contour plot program. *ACM Transactions on Mathematical Software*, 15(1), 1989.
- [13] J. Rasure, D. Argiro, T. Sauer, and C. Williams. A visual language and software development environment for image processing. *International Journal of Imaging Systems and Technology*, 1991.
- [14] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System : A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.
- [15] H. Wright, S.V. Pennington, and G.A. Banecki. Reference model for problem solving in a visual environment. In *Proceedings of Eurographics Workshop on Scientific Visualization*, 1993.