Eclipse and Ellipse: PSEs for EHL Solutions using IRIS Explorer and SCIRun

Christopher Goodyer and Martin Berzins

Computational PDEs Unit, School of Computing, University of Leeds, LS2 9JT, UK
ceg@comp.leeds.ac.uk, martin@comp.leeds.ac.uk
http://www.comp.leeds.ac.uk/cpde/

Abstract. The development of Problem Solving Environments (PSEs) makes it possible to gain extra insight into the solution of numerical problems by integrating the numerical solver and solution visualisation into one package. In this paper we consider building a PSE using IRIS Explorer and SCIRun. The differences in these two PSEs are contrasted and assessed. The problem chosen is the numerically demanding one of elastohydrodynamic lubrication. The usefulness of these packages for present and future use is discussed.

1 Introduction

The field of scientific computing is concerned with using numerical methods to solve real world problems in fields such as engineering, chemistry, fluid flow or biology, which are typically defined by a series of partial differential equations. In solving these problems the ability to use high quality visualisation techniques allows the user to better understand the results generated, to identify any points of interest, or potential difficulties and to obtain greater insight into the solution to a problem more quickly.

This paper will investigate the use of *Problem Solving Environments* (PSEs) to solve a demanding numerical problem in computational engineering. PSEs combine several important stages for the generation of numerical results into one body, thus having synchronous computation and visualisation. There are three ways in which even basic PSEs are advantageous. These are that the input parameters can all be set, or adjusted at run time; the numerical solver is only one part of the PSE and hence it can be possible to change solution methods, if appropriate; and finally the visualisation is an innate component of the package, and results can be visualised and studied as the calculation proceeds. Computational steering gives the PSE another advantage over traditional solution methods because this allows the test problem and/or the solution methods, to be updated during the calculation. The user, thus, "closes the loop" of the interactive visually-driven solution procedure [1].

The PSE construction in this paper has been done in order to compare, contrast and assess the usefulness and the ease of implementation of a challenging engineering problem, in IRIS Explorer [2] and SCIRun [1], two different software packages designed for building PSEs.

The numerical problem selected for integration into a PSE is that of *elastohydrodynamic lubrication* (EHL) in, for example, journal bearings or gears. This mechanical engineering problem requires sophisticated numerical techniques to be applied in order to obtain solutions quickly. An engineer, for example, may want to establish solution profiles for a particular lubricant under certain operating conditions. With a PSE these could be quickly tuned to give the desired results, before tackling, say, a more demanding transient problem. The numerical code for solving EHL problems used in these PSEs is described in detail in [3], and the required changes to its structure will be set out below. Examples of where this extra insight occurs will be given.

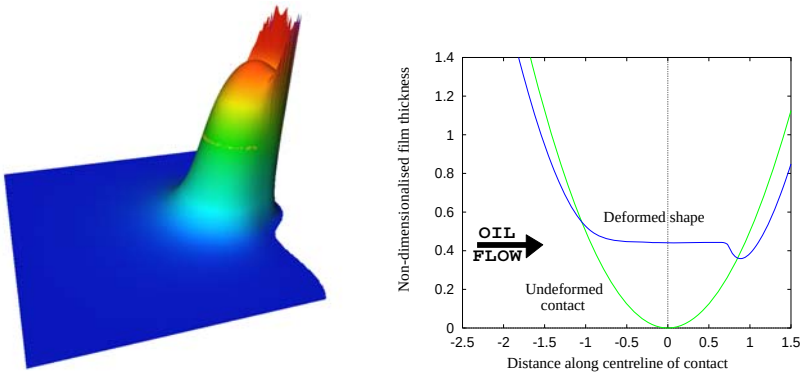
The EHL problem will be described briefly in Section 2 which includes the equation system to be solved, and an outline of the numerical techniques used in the code. Section 3 considers the two PSEs developed. After some general issues have been covered, in Section 3.1 details of the implementation of the EHL PSE into IRIS Explorer, known as ECLIPSE (ElastohydrodynamiC Lubrication Interactive Problem Solving Environment) are given. In Section 3.2 SCIRun is discussed in a similar manner, with the construction of the PSE, known inside SCIRun as ELLIPSE, detailed. In Section 4 the differences between the two systems discussed, both conceptually and in terms of their structure and usefulness as frameworks for building PSEs. Finally, in Section 5 some conclusions are drawn about future development of PSEs bearing in mind the likely needs, in reference to large problem sizes, parallelism and grid-based computations.

2 The Numerical Problem

Elastohydrodynamic lubrication occurs in journal bearings and gears, where, in the presence of a lubricant, at the point of contact there is a very large pressure exerted on a very small area, often up to 3 G Pa. This causes the shape of the contacting surfaces to deform and flatten out at the centre of the contact. There are also significant changes in the behaviour of the lubricant in this area.

A typical solution profile is shown in Figure 1a for the pressure profile across a point contact, such as will be considered in this paper. This is the equivalent situation of a ball bearing travelling along a lubricated plane. With oil flow from left to right it can be seen that in the inlet region there is a very low pressure, in the centre of the contact there is a very high pressure, with an even higher pressure ridge around the back of the contact. Finally, in the outflow a cavitation region is formed where bubbles of air have entered the lubricant, and the pressure is assumed to be ambient there. The corresponding surface geometry profile for the bearing has the undeformed parabolic shape of the contact flattened in the high pressure area. The centreline solution is shown in Figure 1b.

The history of the field is detailed out in papers such as [4]; much information about the numerical techniques currently used to obtain fast, stable solutions is given in both [5] and [3], the latter of which describes in great detail the precise methods used in the code employed in this work.



(a) Pressure solution (b) Film thickness along the centreline

Fig. 1. Typical solutions across an EHL point contact

The EHL system solved depends on many physical parameters concerning both the physical nature of the contacts, the properties of the lubricant used, the loading and the rotation speeds of the surfaces. The solution variables which must be solved for are the pressure profile P, across the domain, the surface geometry H, the viscosity $\overline{\eta}$ and the density $\overline{\rho}$. The full equation system is described in [5] and the appropriate references for the derivations are given therein. The pressure distribution is described by the Reynolds Equation:

$$\frac{\partial}{\partial X}\left(\frac{\overline{\rho}H^3}{\overline{\eta}\lambda}\frac{\partial P}{\partial X}\right)+\frac{\partial}{\partial Y}\left(\frac{\overline{\rho}H^3}{\overline{\eta}\lambda}\frac{\partial P}{\partial Y}\right)-\frac{u_s(T)}{u_s(0)}\frac{\partial(\overline{\rho}H)}{\partial X}-\frac{\partial(\overline{\rho}H)}{\partial T}=0\,, \qquad (1)$$

where u_s is the sum of the surface speeds in the X-direction at non-dimensional time T, λ is a non-dimensional constant, and X and Y are the non-dimensional coordinate directions, The standard non-dimensionalisation means that the contact has unit Hertzian radius, and that the maximum Hertzian pressure is represented by $P=1$. The boundary conditions for pressure are such that $P=0$. For the outflow boundary, once the lubricant has passed through the centre of the contact it will form a free boundary, the *cavitation boundary*, beyond which there is no lubricant. The non-dimensional film thickness, H, is given by:

$$H(X,Y)=H_{00}+\frac{X^2}{2}+\frac{Y^2}{2}+\frac{2}{\pi^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{P(X',Y')\,\mathrm{d}X'\mathrm{d}Y'}{\sqrt{(X-X')^2+(Y-Y')^2}}\,, \qquad (2)$$

where H_{00} is the central offset film thickness, which defines the relative positions of the surfaces if no deformation was to occur. The two parabolic terms represent the undeformed shape of the surface, and the integral defines the deformation of the surface due to the pressure distribution across the entire domain. The conservation law for the applied force (the Force Balance Equation) is given by:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}P(X,Y)\,\mathrm{d}X\mathrm{d}Y=\frac{2\pi}{3}\,. \qquad (3)$$

Since an isothermal, generalised Newtonian lubricant model is being used, only expressions for the density and viscosity will be required. The density model chosen is that of Dowson and Higginson, whilst the viscosity model used is the Roelands pressure-viscosity relation, again see [5].

The solution variables are discretised on a regular mesh. Once discretised, the equations are solved in turn to iteratively produce more accurate solutions. This process is described in detail in [5] where detailed descriptions of the need for, and benefits of multigrid and multilevel multi-integration are given.

The Fortran 77 software used here, *Carmehl*, is described by Goodyer in [3] and includes both variable timestepping for transient calculations [6] and also has the option to use adaptive meshing [7]. Conventionally, once execution is complete then output files of data for the key variables are produced which may then be post processed for visualisation purposes. The user may request that the output solution is saved for continuation purposes on a future run.

3 Problem Solving Environments

This section describes the implementation of the Carmehl code into a PSE form suitable for both IRIS Explorer and SCIRun. The differences between the two products, and their implications are described later in Section 4.

IRIS Explorer and SCIRun have several common ideas: both use a visual programming system where individual *modules* are attached together by a *pipeline* structure, representing the dataflow paths. Each module may have several inputs, either from other modules or from widgets on the control panel of the module, and each represents a separate task which must be performed on the input data. Each module usually produces a new output *dataset*, which is then passed to the next module, or modules, downstream until the results are visualised. Since the datatypes required for visualisation are not the same as those used for numerical calculations conversion modules must be used.

3.1 ECLIPSE in IRIS Explorer

In implementating the EHL code as a module in IRIS Explorer [2] it is possible to build on earlier work employing IRIS Explorer for the development of PSEs, such as Wright *et al.* [8]. IRIS Explorer is marketed by NAG as a "advanced visual programming environment" for "developing customised visualisation applications"[1]. In IRIS Explorer a shared memory arena is used and the pipeline of modules is called a *map*. Although the data can be imagined travelling through the map by the wires, in reality it is only passing pointers to structures of known types in the shared memory arena at the end of each module's execution cycle.

A map in IRIS Explorer executes, normally, by a data set either being read in or generated and then control passes to the next module (or modules) downstream in the map. These in turn execute, provided they have all their required

[1] http://www.nag.co.uk/

inputs and control passes again. If a required input is missing then the module will wait until it is received before executing. If a parallel computer is used, then simultaneous module firings will be done on separate processors. This is because IRIS Explorer starts each module as an entirely separate process in the computer. It will be seen how this has both positive and negative consequences.

The Carmehl code has been implemented as one module containing the entirety of the numerical solver. The module's control panel is used to set the dimensions of the computational domain, the mesh refinement level, along with the total number of iterations required on each execution of the module. Other problem specific properties can also be defined on this control panel including information concerning transient problems, along with parameters governing surface features. The actual non-dimensional parameters governing the case in question may also be set here, or, through the addition of further modules, as shown in Figure 2, the actual operating conditions for the case defined upstream in the map, will be displayed. Once the module has completed execution, the datasets of the pressure and film thickness are sent down the map for visualisation.

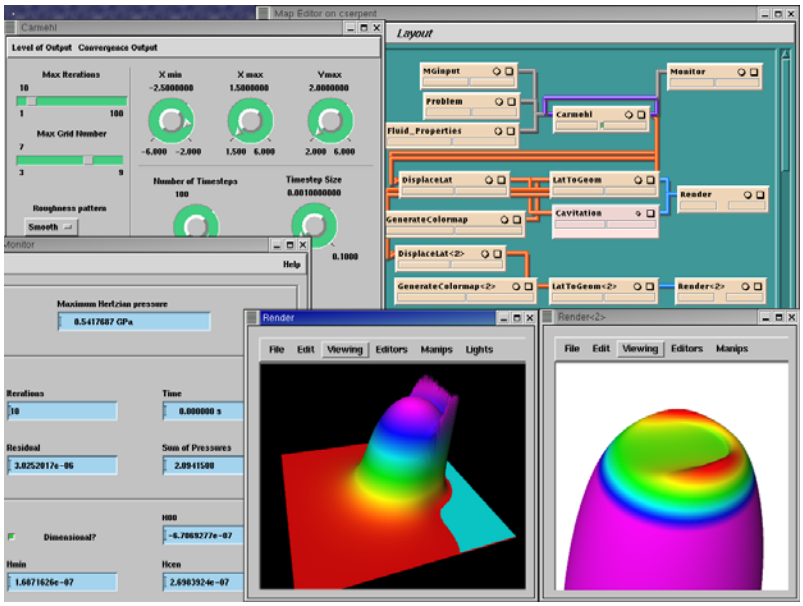


Fig. 2. ECLIPSE running in IRIS Explorer

It is through the addition of extra input modules that steering levels may be abstracted. Shown in Figure 2 are three different input modules which define the physical conditions of the contact, the parameters defining the lubricant, and parameters used to set the number of iterations in the multilevel schemes used. Another input module, not shown here, is that for grid adaptation.

ECLIPSE has been developed from the original Carmehl Fortran code by adding an interface routine written in C. The generation of all the IRIS Explorer data structures and communication is done through the *Application Programming Interface* (API) which is well documented for both C and Fortran. The design of the module control panel is usually done through the Module Builder which allows the widgets to be positioned through a visual interface, rather than by writing code. The Module Builder will also generate the necessary wrapper codes for complete control of the module's firing pattern and communication of data, and these require no alteration by a developer.

Computational steering is implemented in IRIS Explorer using the looping mechanisms provided. Rather than saving results to disk at the end of a run, the work arrays inside Carmehl can be declared as static and hence the previous results are automatically available for use on the next run. A solution used in this manner may provide a good initial estimate for a differently loaded case, or be interpolated for a change of domain size.

The use of the Hyperscribe module [9] would allow another layer of steering to be included. This module stores datasets or variables on disk for future usage, at the user's discretion. If the entire work arrays, previously saved as static, were stored based on the problem's input characteristics then a suite of previously calculated solutions could be created for future invocations of ECLIPSE on separate occasions, or even by other users.

3.2 ELLIPSE in SCIRun

SCIRun has been developed by the SCI group at the University of Utah as a computational workbench for visual programming. Although a longstanding research environment, it has only recently been released as open source software. The discussion below is based on the Version 1.2.0 release[2]. SCIRun was developed originally for calculations in computational medicine [10] but has since been extended to many other applications.

The overall appearance of SCIRun is similar to that of IRIS Explorer, as can be seen in Figure 3 where the implementation of the EHL problem, ELLIPSE, can be seen working. In SCIRun when modules are connected together, they are known as a *network*. The module firing algorithm in SCIRun probes the network from the desired point of firing so that all modules have all the information they need to run, before then sending the information downstream and firing those modules. This means that upstream modules will be fired if they need to supply information to an input port. Similarly all the downstream modules directly affected by the firing will be made aware that new data will be coming.

SCIRun is a multi-threaded program, and hence a single process, with (at least) one thread for each launched module. Therefore every module can have access to all same data without the use of shared memory. This has the advantage that there is more memory available for the generation of datasets to pass between modules, and the disadvantage that any operating system limits on the

[2] SCIRun is available from http://www.sci.utah.edu/

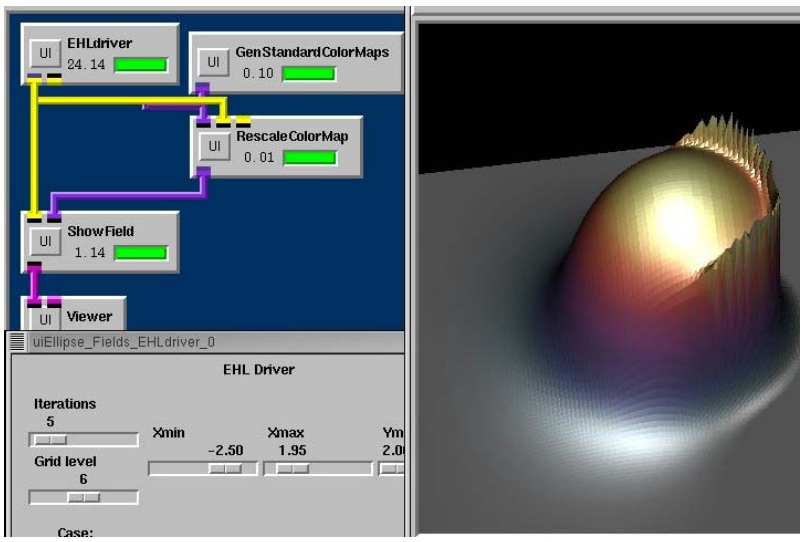


Fig. 3. ELLIPSE running in SCIRun

memory available to a single process apply to the entirety of SCIRun, meaning that calculation and visualisation are all included in the same maximum space allocation defined by the system. It also means that any variables declared as static in one invocation of a module will be the same as used in other invocations, since the operating system cannot differentiate between the two.

SCIRun is written in C++ and requires that at least the driver routine of any contributed module is too. This also means that any of the calls to other SCIRun objects, such as datatypes, needs to be done a SCIRun class. By passing the relevant class pointers to the Fortran, it is possible to return to the class function in order to interact with the SCIRun interface and features.

In Version 1.2.0 all the module control panels, called *UI*s in SCIRun, are written in Tcl and must be programmed by hand. This clearly limits the ease of redesigning the panels, and requires more code to be written to handle the interface between widgets and program variables.

The datatype in SCIRun used to construct meshes for ELLIPSE is *Trisurf*. This is a structure for a surface made up of tesselating triangles. First the list of coordinates of the nodes in the mesh are specified, followed by the connectivities of the points to form the triangles. As yet, SCIRun has no standard modules for manipulating the generated surfaces and so the three dimensional perspective, such as shown in Figure 3, must be included in the creation of the mesh. In order to get the colourmap distributed across the surface then solution values must be stored at each node, again as the mesh is generated.

Since SCIRun is written as a single threaded process it has added flexibility with regard to the rewiring of modules during execution. For the EHL problem, when a transient case is run, the output datasets are prepared and released

down the pipeline for visualisation at the end of each timestep. With more than one solution variable being solved for, there is obviously a choice as to what is visualised at anytime. In SCIRun these changes can be made 'on the fly'. For example if the pressure solution was being visualised, then it is possible to change to a surface geometry plot between timesteps. This is an important feature since it allows the user to learn and experiment interactively.

Parallelism can be easily achieved on SCIRun thanks to its threaded structure. SCIRun has its own implementation of threads that can be easily incorporated into a user's code. The use of threads means a shared memory machine must be used, but within these constraints the parallel performance for numerical calculations is very good. Next generation packages, such as Uintah [11], use a combination of MPI and threads to achieve massively parallel scientific computations on terascale computing platforms.

4 Evaluation

When comparing these packages it is important to remember that IRIS Explorer is a longstanding commercial package supported by NAG whilst the first publically available version of SCIRun was only released last year. The development histories of the two packages are obviously different. IRIS Explorer has a large number of standard modules for reading in various formats of data files, manipulation of datasets, and for the visualisation of this data, with the source of (practically) every module now being part of the distribution. SCIRun was developed in a problem driven way, is completely open source and the number and variety of modules will grow in the coming years.

These different backgrounds are most tellingly reflected in the ease of use of the environments. For a novice use wanting to visualise output data using IRIS Explorer it is a relatively simple process. By addition of a further few modules it is possible to create very intricate output pictures. In SCIRun there is a steep learning curve at present to be able to visualise data, especially with the smaller number of visualisation manipulation modules provided. This manipulation must be done in the initial generation of each mesh and is therefore an additional computational expense for the main module.

In the previous section it was seen how IRIS Explorer and SCIRun have very different paradigms for operation: IRIS Explorer launches each module as a separate process whilst SCIRun is a single, multithreaded process. This has both positive and negative aspects. The advantage of threading is that it is very simple to get data transfer from one module to another. It was seen how SCIRun is more flexible in the rewiring of modules during execution. In IRIS Explorer new connections can only be made after the module has finished executing rather than after each timestep. This is different to the work of Walkley *et al.* [12] where the numerical calculation is less demanding, hence when the control panel is re-read every few timesteps, new connections are registered.

In SCIRun all the modules are loaded as shared libraries. This means that for module developers, coding changes require recompilation of the relevant library,

and currently (though not in future releases) the reloading of SCIRun. In IRIS Explorer only the module which has changed needs to be reloaded using a trivial 'Replace' operation which remembers the data connections of the module.

The design features for constructing a new module in IRIS Explorer benefit from the visual design tools of the Module Builder which allows easy placement of widgets on the control panel and makes the interaction between input variables and those in the driver code very simple. In SCIRun each variable must be captured by the user from the panel, since the module wrapper is generated when the module is first created rather than at compilation, as in IRIS Explorer. This also means that the module wrapper is usually hidden from the developer.

In terms of solving the EHL problem it has been seen that both software packages efficiently handle the PSE structure, achieving similar visualisations by slightly different methods. Using the PSE has been tremendously beneficial in quickly being able to understand complex datasets and see the influence of single parameters. By increasing the regularity of dataset output it is possible to watch the numerical solver converge on the solution. Added insight into the problem was gained using IRIS Explorer's visualisation modules to overlay the surface geometry colour map on the 3D pressure mesh, showing the relationship between pressure and film thickness in a distinctive and hitherto unseen way.

SCIRun benefits from having parallelism at its heart, meaning that incorporating it into an individual module can be accomplished in a relatively straightforward manner. Parallelism in IRIS Explorer has mainly only been done by using the module as a front end to lauching parallel calculations outside of the environment, often on a different machine. Remote processing combined with collaborative visualisation is another area where IRIS Explorer currently takes the lead. In the companion paper [12] Walkley *et al.* show how a similar computational steering IRIS Explorer PSE is first developed for interaction in transient calculations, and then run collaboratively over a network using COVISA [13].

Support for computational steering is central to both packages. Looping of modules is relatively simple to implement in IRIS Explorer through simply wiring relevant modules together, whereas in SCIRun the dataflow mechanism means that care must be taken to ensure that no module is waiting for itself to fire.

5 Conclusions and Future Work

The overall conclusion is that both IRIS Explorer and SCIRun provide good ways to generate PSE environments for problems, such as the EHL problem considered here. An example of how the use of the PSE has enabled extra insight into the problem has been explained.

SCIRun is clearly still in the early stages of its life at Version 1.2.0 whereas IRIS Explorer, now at Version 5.0 is not necessarily too far ahead. Experience suggests that IRIS Explorer's functionality can be recreated in SCIRun, although may involve more programming and a deeper understanding of the software.

It is clear that for the construction of PSEs in the coming years both codes still have work to be done. Parallelism will be very important for increasing

problem sizes. This issue has been successfully addressed in the Uintah PSE already developed in Utah [11]. The issues behind visualisation of significantly larger datasets remain to be fully resolved in IRIS Explorer which can create intermediary copies of the datasets for each module manipulating them.

Acknowledgements

This work was funded by an EPSRC ROPA grant. The Carmehl code has been developed over many years of collaboration with Laurence Scales at Shell Global Solutions. Many thanks are due to Jason Wood for technical assistance with IRIS Explorer and to Chris Moulding for help with SCIRun.

References

1. Parker, S.G., Johnson, C.R.: SCIRun: A scientific programming environment for computational steering. In Meuer, H.W., ed.: Proceedings of Supercomputer '95, New York, Springer-Verlag (1995)
2. Walton, J.P.R.B.: Now you see it – interactive visualisation of large datasets. In Brebbia, C.A., Power, H., eds.: Applications of Supercomputers in Engineering III. Computatational Mechanics Publications / Elsevier Applied Science (1993) 139
3. Goodyer, C.E.: Adaptive Numerical Methods for Elastohydrodynamic Lubrication. PhD thesis, University of Leeds, Leeds, England (2001)
4. Dowson, D., Ehret, P.: Past, present and future studies in elastohydrodynamics. Proceedings of the Institution of Mechanical Engineers Part J, Journal of Engineering Tribology **213** (1999) 317–333
5. Venner, C.H., Lubrecht, A.A.: Multilevel Methods in Lubrication. Elsevier (2000)
6. Goodyer, C.E., Fairlie, R., Berzins, M., Scales, L.E.: Adaptive techniques for elastohydrodynamic lubrication solvers. In Dalmaz *et al.*, G., ed.: Tribology Research: From Model Experiment to Industrial Problem, Proceedings of the 27^{th} Leeds-Lyon Symposium on Tribology, Elsevier (2001)
7. Goodyer, C.E., Fairlie, R., Berzins, M., Scales, L.E.: Adaptive mesh methods for elastohydrodynamic lubrication. In: ECCOMAS CFD, Institute of Mathematics and its Applications (2001) ISBN 0-905-091-12-4.
8. Wright, H., Brodlie, K.W., David, T.: Navigating high-dimensional spaces to support design steering. In: VIS 2000, IEEE (2000) 291–296
9. Wright, H., Walton, J.P.R.B.: HyperScribe: A data management facility for the dataflow visualisation pipeline. Technical Report IETR/4, NAG (1996)
10. Johnson, C.R., Parker, S.G.: Applications in computational medicine using SCIRun: A computational steering programming environment. In Meuer, H.W., ed.: Proceedings of Supercomputer '95, New York, Springer-Verlag (1995) 2–19
11. de St. Germain, D., McCorquodale, J., Parker, S., Johnson, C.R.: Uintah: A massively parallel problem solving environment. In: Ninth IEEE International Symposium on High Performance and Distributed Computing. (2000)
12. Walkley, M.A., Wood, J., Brodlie, K.W.: A collaborative problem solving environment in IRIS Explorer. Submitted to: Proceedings of the International Conference on Computational Science 2002. (2002)
13. Wood, J., Wright, H., Brodlie, K.W.: Collaborative visualization. In: Proceedings of IEEE Visualization 97. (1997) 253–259