Practical Tools for Computational Steering.

S. M. Pickles, R. Haines, R. L. Pinning, and **A. R. Porter**

Manchester Computing,
University of Manchester,
Oxford Road,
Manchester, M13 9PL, UK.

Computational steering refers to the real-time interaction of a scientist with their running simulation code. Despite the many benefits associated with computational steering, its uptake to date has been limited. In this paper we discuss the reasons for this and how the computational steering library and associated tools developed as part of the RealityGrid project aim to tackle them. We describe the functionality of the steering library and the use of Grid services in constructing a generic, dynamic architecture for discovering, steering and connecting visualization software to running simulations. The use of on-line visualization for providing feedback to the scientist is described, including the ways in which it may be enhanced through tools such as Chromium and Access Grid. Finally, we illustrate the flexibility of our approach by describing the functionality that has been added to various simulation codes as part of the RealityGrid project.

I. INTRODUCTION

Computational steering [1] can be applied to good effect in a variety of computational disciplines. By monitoring the progress of simulations, aided by on-line visualization, the computational scientist avoids losing cycles to redundant computation or even doing the wrong calculation. By tuning the value of steerable parameters, the computational scientist quickly learns how the simulation responds to perturbations and can use this insight to design subsequent computational experiments. Computational steering can even help algorithm development as the granularity of control it confers tends to be better suited to the physical system under study than the line-by-line control offered by a source-level debugger. However, despite the obvious advantages of computational steering, its routine application is still the exception rather than the rule.

In this paper we review a number of factors that have retarded the widespread adoption of computational steering techniques, and discuss how we have addressed these factors in the RealityGrid project. [2] We describe the RealityGrid computational steering library, and how it can be used to instrument a code for steering with only modest effort on the part of the application developer. We describe how simulations can be exposed as Grid services and the flexibility that this confers. The support provided by the steering library for implementing on-line visualization is discussed, along with the use of tools to improve visualization performance and allow collaboration between scientists at different institutions. We conclude with a brief overview of successful applications of our techniques.

II. COMPUTATIONAL STEERING

There exist a number of different approaches to the design and implementation of a computational steering system. A first step is often a system that is built around a single, pre-existing application and is intended to fulfil a specific purpose. This has the advantage that it is typically developed by the application scientists themselves and therefore is maintained by them and retained as the code is developed. However, such a scheme is unlikely to be of use to other projects.

An alternative approach is to embed the application as a module within a more general Problem Solving Environment (PSE), as typified by the SCIRun framework. [3] Although these environments provide powerful tools for interacting with an application, the work required to extend an existing application code for use in such a PSE has been a significant barrier. In addition, such PSEs have typically been developed with the intention of running all of the modules on a single machine, thus limiting the scale of the scientific computation that can be tackled. However, with the advent of the Grid concept some PSEs (such as Uin-

tah [3]) are being extended so that modules may be distributed over available resources.

Computational steering is typically closely associated with the ability to monitor the state of a running simulation via an on-line visualization. It is therefore not surprising that many steering systems have been developed as part of a visualization package. Examples of such work include CUMULVS [4], VISIT [5] and extensions to IRIS Explorer. [6] Although this solution makes use of existing, powerful visualization tools, it does tie the user to them and also to a machine capable of running them.

Another approach to computational steering that is popular because of its ease of access is that of providing a web-based interface to a running application. This method is typified by the Cactus framework [7] in which a scientist may use a web client to connect to a web-server linked to their running simulation. Although good for simple steering, the visualization that can be performed through a web client is limited and therefore this is normally performed by a separate application.

We have taken a component-based approach to the development of a computational steering system within the RealityGrid project. By explicitly separating the role of steering from that of online visualization it becomes possible to produce clients tailored to a variety of platforms – from PDAs to laptops to workstations running MS Windows. Such an approach does not preclude the creation of a client which performs both visualization and steering, possibly within a modular visualization environment with which the application scientist is already familiar.

Within RealityGrid we are concerned with making pre-existing simulation codes steerable. These codes are written in a variety of languages, and may be serial or parallel. These are living codes, undergoing active development and optimisation. When making a code steerable it is not acceptable to compromise on performance, or to demand a wholesale re-factoring of the code, or to cause a bifurcation of the source tree into steerable and non-steerable versions. These considerations led us to choose an approach in which the code is instrumented for steering by inserting calls to a library with a well-defined API. If the instrumented source code is to be proof against future evolution of the infrastructure, it is important

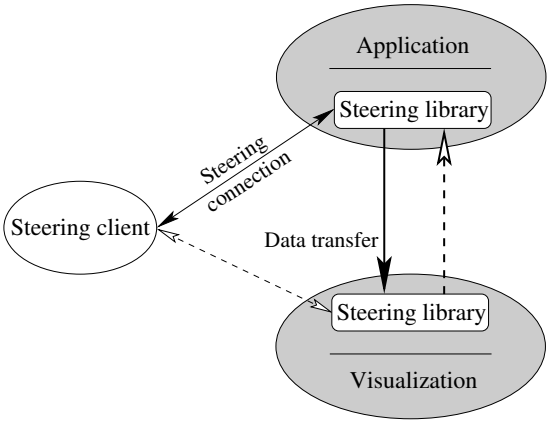

FIG. 1: Basic architecture for RealityGrid steering. Each ellipse represents a distinct application, the location of which is not constrained. The client can dynamically attach to the running application, perform some steering activity and then detach again. The application may also use the steering library to send data sets to some other component, *e.g.* a visualization package.

that the API permit multiple implementations and be a candidate for standardisation; consequently the API should hide all implementation-specific details from the application, exposing only those things which are meaningful to the application developer.

III. ARCHITECTURE OF STEERING

It is natural, especially for large-scale computational science, to distribute the computation, visualization and steering client over a Grid. (This approach generalises well to the case where one has two or more computationally-intensive components forming a coupled simulation.) A simple illustration of the distributed architecture that we have followed in developing our steering software is shown in Figure 1. This shows a typical configuration in which an application periodically emits data sets which are consumed by a down-stream visualization component.

The mechanism used for the transfer of messages between the steering client and the steered application and that used for the transfer of data between the application and visualization need not be the same and are transparent to the application developer. Currently the RealityGrid steering library supports file-based transfer and SOAP over http (within the Open Grid Services Architecture

or OGSA) for steering communication. In both cases, the same XML messages are transferred between client and application, however, in the former method they are written to (and read from) file while in the latter they are sent as SOAP packets in calls to a Grid service. For data set transfer, the library contains both file-based and socket-based implementations.

The inclusion of file-based implementations for both types of communication makes getting an initial working steering set-up simpler since it avoids the complications introduced by networking and reduces dependencies on external software. Since the steering API is independent of implementation, a scientist may concentrate on instrumenting their application for steering, using file-based communications in a local environment. Once the correctness of the code is established, moving to a distributed environment will not require any changes to the application source code.

Figure 2 illustrates our rendering of computational steering in the OGSA. The middle tier, implemented in OGSI::Lite [8], enables discovery of running applications and dynamic connection/disconnection of components, and mediates communications in a firewall-friendly way. By exposing the steering controls as operations of a Grid service, the protocol of steering is encapsulated in WSDL. This has enabled other groups to develop steering clients in Java (as used in the RealityGrid Web Portal by M. Egbert of EPCC) and in .NET (as used in the Windows and PDA versions of the client by S. Nee of Loughborough [9]). Our architecture is component friendly, which has enabled the integration of RealityGrid-compliant steerable applications into the ICENI [10] component framework.

The dynamic connection/disconnection of components is dealt with when a component is launched. For instance, when a user chooses to launch a visualization, the launching software discovers what components are already running and asks the user to choose which one to use as a data source. It then creates the Steering Grid Service (SGS) for the visualization component and initialises it with the identity of the SGS to be used as a data source. At run time, the visualization (via the steering library) queries the visualization SGS for the end point of the data connection. In turn, the visualization SGS queries the SGS acting as a data source and obtains the end point for the connection. This information is passed back to the steering library and the connection is made.

IV. THE REALITYGRID COMPUTATIONAL STEERING LIBRARY

Since the majority of scientific codes (both new and existing) are written in either Fortran, C or C++ and run on a wide variety of platforms we chose to implement the steering library API in C. A complete set of Fortran90 (F90) bindings is provided. We took the decision to avoid including explicit support for parallel codes within the library. This means that *any* serial or parallel application may use the steering library, no matter what paradigm or framework it uses.

The steering library consists of two parts: an application side and a client side. The client is typically a user in control of a steering client. On the application side, a variety of features are supported. These include the facility for the application to register both monitored (read-only) and steerable (changed only through user interaction) parameters. The library supports a set of pre-defined commands: ‘pause’, ‘resume’, ‘detach’ and ‘stop.’ In addition to these pre-defined commands, the library allows the user to instruct the application to emit or consume any data sets that it has previously registered. Similarly, the client may instruct the application to take a checkpoint, after which it should call the library to register the checkpoint’s existence. Finally, provided that the application supports it, the client may also instruct the application to restart from a checkpoint which has been recorded previously.

The latter functionality is particularly important since it provides the basis of a system that allows the scientist to ‘rewind’ a simulation (by restarting from a previous checkpoint). Having done so, it can then be run again, perhaps after having steered some parameter or altered the frequency with which data from the simulation is recorded. This procedure results in a tree of checkpoints which, when combined with a log of all steering activity, provides a good way of recording the exploration of the parameter space of a simulated system. The GRASPARC project [11] is an example of another system with this functionality.

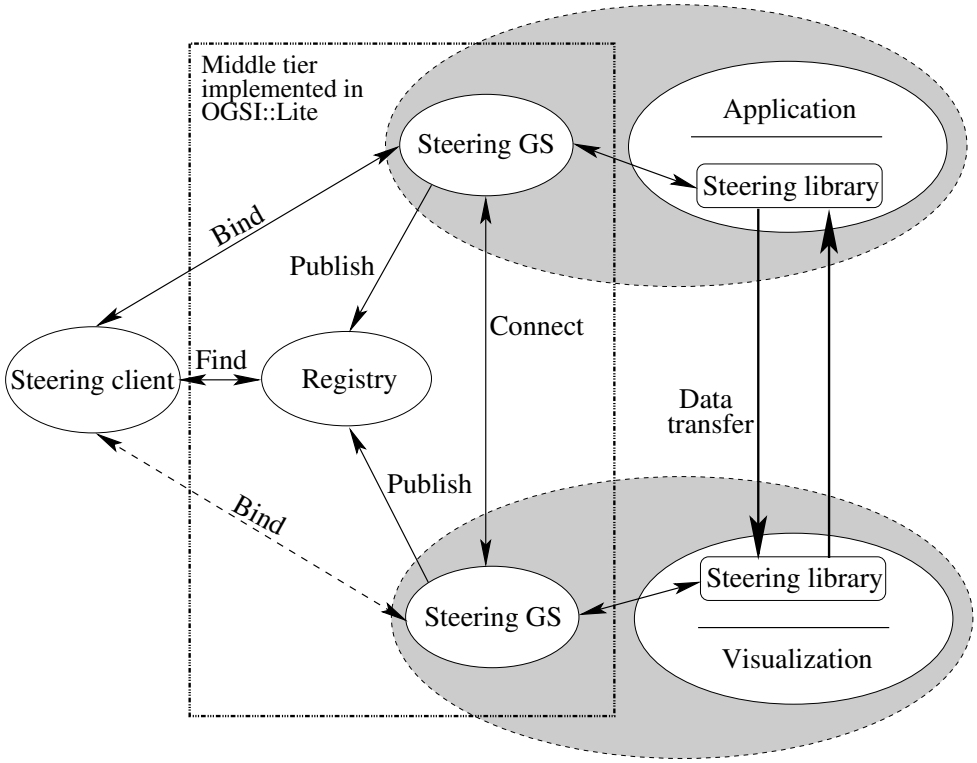

FIG. 2: Architecture for RealityGrid steering within the OGSA. The application and client communicate by exchanging messages through intermediate grid services. The Steering Grid Service (SGS) provides the public interface through which clients can steer the associated application as well as providing support for constructing distributed (multi-component) applications.

In order to maximise the flexibility of the library, we use a system of 'reverse communication' with the application. This means that, for most actions, the library simply notifies the application of what it needs to do. It is then the application's responsibility to carry out the task, possibly using utility routines from the steering library. This allows the application scientist to decide how much steering functionality he wishes to implement and lowers the barrier to making a code steerable.

For example, in order to implement the bare minimum that is required to make a code steerable, just five calls to routines in the steering library are required. (Two for enabling and initialisation, one to register a variable as steerable, one within the main loop to check for messages from a steering client and one to clean up when done.) The intention is that the application scientist should be able to make these additions and therefore will retain control of the code and the ability to maintain it.

The client side of the library is intended to be used in the construction of a steering client. We have developed a generic steering client using C++ and Qt (a GUI toolkit) [12] which is capable of steering *any* application that has been 'steering enabled' using the library. The commands supported by an application and its monitored/steered parameters *etc.* are discovered as part of the connection process and the client GUI is then populated accordingly. The client can show plots (updated in real time) of the history of one or more monitored parameters. It also provides checkpointing control, enabling the user to request that the application take a checkpoint (provided it supports such an action) or restart from an existing checkpoint — the user is able to view a snapshot of all parameter values for any logged checkpoint.

V. ON-LINE VISUALIZATION

The effectiveness of computational steering can be significantly enhanced by providing visual feedback to the scientist. Every time a parameter is adjusted, the computational scientist needs to understand how the behaviour of the simulation

is affected. Many codes allow a scientist to perform a calculation and then visualize the results in a separate (maybe bespoke) application, but this is not a natural way of working when using computational steering. If the user is changing parameters while the code is running, then it is an advantage for them to see what effects those changes have immediately. The RealityGrid steering library allows the user to connect a visualization application to a running code and visualize the current state of the simulation at any time — typically every few time-steps.

To take advantage of on-line visualization the user needs to do two things:

- Instrument the simulation code so that it can emit data samples on request.
- Instrument the visualization application so that it can accept data from the steering library.

In many parallel applications, the data samples to be emitted will be decomposed over multiple nodes and might be so large as to preclude their reconstruction on a single node for the purposes of IO. The steering library is therefore designed around the concept of data samples being emitted in 'slices.' Consequently, instrumenting a parallel simulation code will often involve adding code to gather the data that is to be emitted, in addition to adding the necessary calls to the steering library. It is left to the user to decide what the slices should consist of and hence, how to reconstruct the data set in the application (typically a visualization) that is to receive the data.

Once the necessary instrumentation has been added to the simulation and visualization, the steering library then manages the communications between the two processes (see Figures 1 and 2). The majority of the visualization applications that have been developed so far within the project have used vtk [13]; the steering library is used in the implementation of a reader for data of the format emitted by the application.

During the live TeraGyroid experiments at SuperComputing03 [14, 15] the visualizations were produced on a Linux cluster using Chromium [16] and were distributed to audiences over the Access Grid. [17] Chromium is a system for distributing a visualization task across a cluster of graphics workstations or a number of graphics pipes without altering the original code [18]. A major constraint that the video encoders used by Access Grid impose is that the resolution of a video stream must be 352×288 pixels and the compression algorithms used are lossy. Chromium can help to overcome this problem by allowing the user to split the output visualization into a number of tiles. Each of these tiles can be rendered on a separate machine (to potentially improve performance on more complex visualizations) and then broadcast down a separate Access Grid stream. These separate streams can then be tiled back together at each Access Grid node, either manually by the node operator or automatically by an enhanced version of the Access Grid software. The resultant visualization is improved in three ways:

- The image is bigger. People sitting in an Access Grid node are typically sat some distance away from the screen.
- The image has a higher resolution (in multiples of the Access Grid stream resolution). Finer detail can be represented accurately.
- Artifacts produced by the compression algorithm become less obtrusive and are less likely to obscure fine detail.

An alternative system for distributing visualization results is OpenGL Vizserver [19]. Vizserver allows a visualization application to be run remotely on SGI hardware, with the resulting visualization broadcast to any number of clients. These clients can each interact with the visualization as if it were running on the local machine, therefore providing a level of collaboration as yet not available using the Access Grid alone.

Using a combination of these tools, the user can get immediate visual feedback from a running simulation and collaborate with colleagues around the world in deciding how each simulation should be directed. The TeraGyroid experiment showed that this method of working could produce results very quickly due to the direction provided by online visualization.

VI. STEERING-ENABLED APPLICATIONS

In this section we describe the steering functionality that has been added to six different

physics-based applications.

A. LB3D — a Lattice Boltzmann code

LB3D is a parallel (MPI-based), F90 code for performing three-dimensional Lattice-Boltzmann simulations. [20] Extensive steering instrumentation has been added which allows the user to steer all parameters of the simulation including coupling constants, fluid densities, relaxation times and data dumping frequencies. Steerable data dumping frequencies enable the user to increase the amount of generated data during periods of the simulation when events of particular interest are happening. This helps to save on the use of expensive/finite disk space. LB3D also has the ability to create a checkpoint or restart from an existing checkpoint when requested by a steering client. The checkpoints it produces are both *malleable* (a checkpoint created by a job running on n processors may be used to start a job on m processors where $n \neq m$) and portable across heterogeneous architectures.

This functionality has made it possible for us to build a system that enables running jobs to be migrated from one resource to another, as used during the TeraGyroid experiment at SuperComputing03. [14] At present this is initiated by user request but ultimately we aim for additional migration management using automated performance control software. [21]

On-line visualization for LB3D is provided by a vtk-based application which obtains data from the simulation via the steering library.

B. Ludwig — a Lattice Boltzmann code

Ludwig, from the University of Edinburgh, is a second MPI-based, F90 code for performing Lattice-Boltzmann simulations. It has had basic steering functionality (control of certain simulation parameters) added to it. Previously, online visualization has been provided by sending data to a bespoke module in AVS/Express. However, we are investigating the use of the steering library/vtk-based viewer that is currently used with LB3D.

C. LBOMD — classical molecular dynamics

LBOMD, from Loughborough University, is an MPI-based, F90 code for simulating nano-indentation and collision-cascade processes within solids. The code is instrumented so that a steering client may be used to monitor parameters such as the current simulation time, the total energy of the system, the type of thermostat, the system temperature and the time of the next IO (failsafe file or data for visualization). Through a steering client, a user is also able to control various aspects of the simulation including how frequently the simulation writes out data, the temperature setting of the thermostat and (for nano-indentation) the maximum indentation depth. [22] On-line visualisation is performed using a bespoke application built upon vtk and the computational steering library.

D. NAMD/VMD — classical bio-molecular dynamics

NAMD [23] was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. It is a parallel C++ code designed for performing large-scale molecular dynamics simulations using classical force-fields. VMD is a visualization package (also written in C++) designed to interface with NAMD (via a socket connection) and allows the user to monitor properties such as timestep number, temperature *etc*. It also enables the user to perform Steered Molecular Dynamics by allowing them to apply forces to the molecule being simulated. These forces are sent back to NAMD and incorporated into the simulation.

In order to extend the steering functionality of NAMD we have instrumented both it and VMD using the steering library such that they take the places of the 'Application' and 'Visualization' of Figure 1, respectively. In this way, the existing steering functionality of VMD is retained while additional steering functionality can be accessed via any one of the steering clients developed within the project. By extending NAMD's existing checkpointing facilities to allow them to be controlled via a steering client, we aim to introduce the job migration and spawning facilities

that LB3D currently supports.

E. POLYSTEER - Monte Carlo polymer simulation

POLYSTEER (developed by D. Mason at the University of Oxford) is an MPI-based, F90 code used to simulate the properties of polystyrene molecules. It has been instrumented such that a user may adjust the (many) parameters controlling the Monte Carlo moves while monitoring the efficiency with which the simulation explores its configuration space. This enables the scientist to gain insight into the behaviour of the polymer while improving the efficiency of the simulation. Online visualisation of the polymer configurations is provided by a Java2D-based application.

F. Navier-Stokes turbulence, vortex dynamics and knot theory codes

B. Boghosian's group at Tufts University has developed three, 3D parallel Navier-Stokes solvers: (i) a multiple-relaxation time Lattice-Boltzmann solver; (ii) an entropic Lattice-Boltzmann solver and (iii) a pseudospectral solver for the study of turbulence in single phase fluids. All three codes feature vtk graphical output and both the simulation and graphics codes have been instrumented to use the RealityGrid steering library. This group has also developed a telnet-based steering client for interacting with these codes.

VII. CONCLUSIONS

We have described the RealityGrid computational steering library and associated tools and their use in bringing the advantages of computational steering to bear in a variety of codes and simulation scenarios. The relatively small amount of coding that is required to achieve basic steering functionality has been emphasised. This is in keeping with the philosophy that this technology will only achieve widespread adoption if the scientists developing the codes are happy to take this work on themselves.

The flexibility conferred by representing components (simulations, visualizations, *etc.*) as Grid services has been outlined. We have described the architecture that we have used in building a framework upon these services which allows visualizations to be dynamically connected to running simulations and steering clients to discover and connect to simulations.

The RealityGrid steering library, steering client and associated OGSI::Lite-based middleware are all available for download under an open-source licence from `http://www.sve.man.ac.uk/Research/AtoZ/RealityGrid/`.

VIII. ACKNOWLEDGEMENTS

Financial support for this work was provided by the Engineering and Physical Sciences Research Council through RealityGrid grant number GR/R67699.

[1] J. D. Mulder, J. J. van Wijk, and R. van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15:119–129, 1999.

[2] The RealityGrid project: `http://www.realitygrid.org`.

[3] C. Johnson, S. Parker, D. Weinstein, and S. Heffernan. Component-based, problem-solving environments for large-scale scientific computing. *Concurrency and Computation: Practice and Experience*, 14:1337–1349, 2002.

[4] P. M. Papadopoulos and Kohl J. A. Dynamic visualization and steering using PVM and MPI. *Lecture Notes in Computer Science*, 1497:297–303, 1998.

[5] T. Eickermann, W. Frings, and A. H"aming. *Visit - a Visualization Interface Toolkit*. Research Centre Juelich GmbH, 2002. `http://www.fz-juelich.de/zam/visit/`.

[6] K. Brodlie, S. Mason, M. Thompson, M. Walkley, and J. Wood. Reacting to a crisis: benefits of collaborative visualization and computational steering in a grid environment. In *paper presented at the UK e-Science All Hands Conference, Sheffield*, 2002.

[7] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf. The cactus framework and toolkit: Design and applications - invited talk. *Lecture Notes in Computer Science*, 2565:197–227, 2003.

[8] M. Mc Keown. *OGSI::Lite*. A Perl implementation of an OGSI-compliant Grid Services Container. http://www.sve.man.ac.uk/Research/AtoZ/ILCT.

[9] S. P. Nee and R. S. Kalawsky. Developing a Roaming PDA-based Interface for a Steering Client for OGSI::Lite using .Net: Practical Lessons Learned. In *paper presented at the UK e-Science All Hands Conference, Nottingham*, 2004.

[10] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington. ICENI: An open grid service architecture implemented with jini. In *paper presented at SuperComputing 2002, Baltimore, USA*, 2002.

[11] K. W. Brodlie, L. A. Brankin, G. A. Banecki, A. Gay, A. Poon, and H. Wright. GRASPARC: A problem solving environment integrating computation and visualization. In G. M. Nielson and D. Bergeron, editors, *Proceedings of IEEE Visualization 93 Conference*, page 102. IEEE Computer Society Press, 1993.

[12] Trolltech Qt - A GUI Toolkit, http://www.trolltech.com.

[13] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Kitware, Inc., 3rd edition, 2003.

[14] S. M. Pickles, R. J. Blake, B. M. Boghosian, J. M. Brooke, J. Chin, P. E. L. Clarke, P. V. Coveney, R. Haines, J. Harting, M. Harvey, S. Jha, M. A. S. Jones, Mc Keown M., R. L. Pinning, A. R. Porter, K. Roy, and M. Riding. The TeraGyroid Experiment. In *Workshop on Case Studies on Grid Applications, held in conjunction with GGF10, March 2004, Berlin, Germany*, 2004.

[15] J. Chin, P. V. Coveney, and J. Harting. The TeraGyroid project — collaborative steering and visualization in an HPC Grid for modelling complex fluids. In *paper presented at the UK e-Science All Hands Conference, Nottingham*, 2004.

[16] Chromium Homepage: http://chromium.sourceforge.net.

[17] The Access Grid project: http://www.accessgrid.org.

[18] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A stream-processing framework for interactive rendering on clusters. In *ACM Transactions on Graphics*, pages 693 – 702. SIGGRAPH 2002, ACM Inc., 2002. Also available from: http://graphics.stanford.edu/papers/cr/.

[19] Silicon Graphics, Inc. *OpenGL Vizserver™ 3.1: Application-Transparent Remote Interactive Visualization and Collaboration - White Paper*. http://www.sgi.com/pdfs/3263.pdf.

[20] J. Chin, J. Harting, S. Jha, P. V. Coveney, A. R. Porter, and S. M. Pickles. Steering in computational science: mesoscale modelling and simulation. *Contemporary Physics*, 44(2):18, 2003.

[21] K. Mayes, G. Riley, R. W. Ford, M. Lujan, and T. L. Freeman. The design of a performance steering system for component-based grid applications. In V. Getov, M. Gerndt, A. Hoisie, A. Maloney, and B. Miller, editors, *Performance Analysis and Grid Computing*, pages 111–127. Kluwer Academic Publishers, 2003.

[22] C. F. Sanz-Navarro, S. D. Kenny, A. R. Porter, and S. M. Pickles. Real-time visualization and compuatational steering of molecular dynamics simulations of materials science. In *paper presented at the UK e-Science All Hands Conference, Nottingham*, 2004.

[23] L. Kal, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, 151:283–312, 1999.