Retrospective checkpointing for efficient management of computational steering

William Pinnington and John Brooke

School of Computer Science, Kilburn Building,
The University of Manchester, Oxford Road, Manchester, M13 9PL.

Abstract

The investigation of the behaviour of complex numerically simulated systems in response to key control parameters is an important topic in e-Science. One method of investigating such systems is via computational steering, where a scientist or group of scientists changes (steers) the value of control parameters as the simulation is evolving. If such simulations are run over multiple sessions they create complex data structures which can be described as steering "trees". In this paper we investigate a proposed method for reducing the cost of managing steering trees and describe a reference model simulation which offers the potential of comparing the efficiency of different methods of managing such steering trees.

1 Introduction

Computer hardware and software have played an important role in the advancement of mankind's knowledge of the physical world since their widespread adoption in the middle of the last century. Computers running simulation software allow scientists to better understand complex physical systems, from large scale problems such as forecasting the weather [2, 15] to small-scale problems such as the flow of tomato ketchup from a bottle [8, 3]. By convention [15, 14, 5], this software is referred to as *the simulation* and the user of the simulation is referred to as *the scientist*. In modelling a real system on a computer it is still the case that key physical processes are modelled via parameters rather than directly, since the information processing capacity of computers is still many orders of magnitude less that the full information content of the real system. Such parameterisation predates the computer age in the form of modelling assumptions captured as dimensionless parameters. Examples are the Rayleigh number which compares the effect of advection with diffusion in a fluid system, the Prandtl number which compares the magnitude of heat diffusion to fluid flow diffusion. Full modelling of such effects would require the accurate simulation of atomistic processes, hence their presentation as single numbers represents a modelling assumption to make the system solvable. As an example of the problems facing a full modelling of fluid flow, a microscopic model of a 1mm^3 of monatomic gas, at room temperature, would constitute around 10^{17} variables [7], requiring 400 petabytes of storage for a single instantaneous state of the system alone. Calculations using Moore's law[1] determine that the computational power required to fully model fluids is still a couple of decades away [9].

Thus, we often have the situation that such parameters are insufficiently well known and that changes in their value can have large effects on the behaviour of the simulated system. An illustrative example is a parameter which represents the degree to which two different fluids (e.g. oil and water) can mix [10]. By adding chemicals known as surfactants to the mixture this miscibility is altered and the fluid structures can change dramatically. Such structural changes are known as bifurcations, indicating that on changing the control parameter new solutions can develop (bifurcate) from previous ones. Thus investigating the response of simulations to changes in control parameters is very important, either because we wish to simulate physical experiments (e.g. changing the misci-

[1] Moore's law refers to the statement made by Gordon Moore, the founder of Intel, in 1964. He observed that because of improvements in manufacturing, engineers would be able to double the number of transistors on a chip every eighteen months [17]. This roughly equates to a doubling of computational power ever year and a half and this has held true for over four decades.

bility of a mixed fluid) or because we wish to investigate the natural variability of the system resulting from our imperfect knowledge of the processes represented by the parameter (this is important in weather prediction for example).

Two main approaches have developed to the computer simulation of responses to changes in control parameters. In the ensemble approach many different simulations are launched with different values of the control parameter and the runs are examined *a posteori*. In the computational steering approach the control parameters are varied as the simulation is being performed. Each approach has benefits, the ensemble approach allows for a statistically valid estimate of natural variability in terms of likelihood estimates. The steering approach can better mimic the changing of control parameters by experimental procedures or to enable a careful structured investigation of a bifurcation. These approaches are very different in terms of the interaction of the *scientist* with the *simulation*. In this context, computational steering can be described in terms of an interaction between a human and a machine bringing a level of complexity in management of the simulation that does not exist in the ensemble approach.

Computational steering has been claimed to make more efficient use of computational resources by enabling the scientist to monitor the progress of simulations with the aid of on-line visualisation. This, coupled with a developed scientific intuition, allows the computational scientist to avoid performing redundant computations [14]. However the result of multiple changes in parameters creates potentially complex tree structures where system state and control parameters are linked together. The history of parameter changes and the states that such changes produce can lead to problems of management of the simulation state which can remove the benefits of steering unless tools are provided for their efficient management.

The implementation and architecture of a computational steering system is subject to a number of alternative approaches ranging from close tie-in to the existing simulation, to a more general Problem Solving Environment, e.g. SCIRun [12], to the component-based approach which explicitly separates the simulation, computational steering and visualisation [7], e.g. RealityGrid. The RealityGrid client software steers pre-existing simulation code via either a command line interface (CLI) or a graphical user interface (GUI) depending on the needs of the scientist. Checkpointing enables the scientist

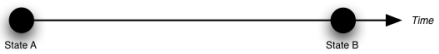

Figure 1: Simulation progress between states.

to *store* the state of a simulation; a checkpoint is written to disk and this can be used to restore and continue the simulation at another time [1]. A checkpoint can be created manually, or a schedule can automatically create a checkpoint at pre-defined intervals. Checkpointing currently supports error recovery and job migration but could also enable more efficient techniques for exploring parameter space, e.g. perform what-if scenarios and re-perform a simulation at a different resolution. When multiple checkpoints are stored, a history of saved states is created and any of these checkpoints can be used to restart the simulation. Experimentation with steered parameters leads to the formation of a tree of checkpoints [4], where different branches can be explored in parallel [13].

2 Managing Checkpoint Trees

We need to distinguish between the state of the simulated system (which may require large amounts of storage) and the record of the changes in parameter values which requires far less storage. However, in practice, these are intimately linked. Suppose a scientist is running a simulation that develops from State A to State B over time (see Figure 1). Time is depicted as a line that proceeds from left to right and states are nodes represented by circles.

For the state of the simulation at the nodes to be precisely definable after the simulation code has been executed, it is necessary to checkpoint the simulation at each node. In checkpointing the simulation, the state of the simulation is saved to disk and the checkpoint contains the parameters necessary for the simulation to be re-started from the moment the checkpoint was created. The client software instructs the simulation code to create and load checkpoints in addition to resuming the simulation from a checkpoint. So far we assume that all control parameters are held constant in the evolution AB.

In Figure 2, Checkpoint B contains more than just the state of the simulation at node State B; it also contains a reference to the parent checkpoint, Checkpoint A, and the control parameter values in the evolution AB. If these were changed then A might evolve to a different state

Figure 2: Checkpoints save the state of the simulation and are represented as cubes of data.

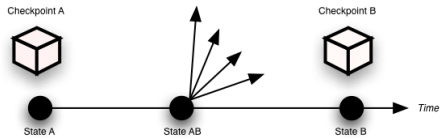

Figure 3: Multiple resumptions from an unsaved state.

B*. In this way, a checkpoint tree is created; this tree can be navigated and the simulation can be restarted from any checkpoint the scientist chooses. However, the complexity of the calculations being performed by the simulation demands significant disk space to be made available. When the scientist instructs the simulation to create a checkpoint, the output potentially requires tens, or hundreds of gigabytes of disk space [14, 11]. Hence, the cost associated with saving checkpoints to disk is substantial.

Given the significant costs associated with checkpointing, checkpoints are typically only created when it is essential to do so. This means that they are often created to support recovery from failure rather than supporting the exploration of the response of the system to changes in paramenter space. To see the distinction, we now allow a single control parameter to change as we evolve from A to B and on arriving at B it is observed that a major change in the state of the system has occurred. In our mixed fluid example, the fluids may appear completely mixed at A but have separated into geometrically defined structures at B. Somewhere in the trajectory we have encountered a critical point and the whole neighbouring region is of fundamental importance for the science of the simulation.

Thus, it is useful to supplement ad hoc checkpoint creation with the automated the creation of checkpoints at predetermined intervals and this has already been achieved. For example, simulations can be set with a predetermined measuring frequency, e.g., a typical lattice-Boltzmann simulation might be 20,000 timesteps in length with a checkpointing frequency of 100 timesteps [7]. However, periodic checkpointing in this fashion is potentially wasteful and cannot ensure that there is a checkpoint that immediately precedes a bifurcation. Given that checkpoints are costly, it is possible that a substantial amount of elapsed time between saved states—significant in relation to the efficient computational study of the bifurcation.

As a scientist gains experience with the simulation, it might be assumed that he/she would learn to take efficient checkpoints by balancing the cost and predicting when a bifurcation is likely to occur. However, this assumes that the system is, to some extent, understood and does not apply to the more explorative, collaborative and serendipitous studies of complex systems [5]. The use case we envisage is of a team of collaborating scientists following different trajectories simulataneously in a collaboration extended over several sessions. We also look towards semi-automation of the steering process, the change in structure from A to B may occur in a long overnight run on a supercomputer and may not be detected until it has developed sufficiently for some monitoring routine to detect.

The limitation of restarts only from discrete, pre-defined moments in time has concomitant implications for efficiency, especially when the computation from A to B is expensive in terms of processing power and the storage. Take the scenario in Figure 3 as an illustration of this inefficiency. As a very simple example, if there are 10^6 timesteps between node State A and node State B and the scientific team desires to run 10^2 what-if scenarios from the simulation's state half-way between these nodes, then the simulation will execute 5×10^7 unnecessary timesteps which is an inefficient and wasteful use of expensive resource. These figures are not at all unrealistic when considering simulation projects such as the Virtual Physiological Human [16], since the steering trees involved may become exponentially complex. Consequently, we must develop efficient processes to automate the efficient management of state [5].

3 Economising computing cycles

As a first step, we examine the situation described in Figure 3 where a bifurcation exists on the trajectory between node State A and node State B. In the abscence of any other knowledge we can employ a bisection algorithm to search for the bifurcation point, requiring a checkpoint to be retrospectively created, we call this AB. With Checkpoint AB at the scientists' disposal,

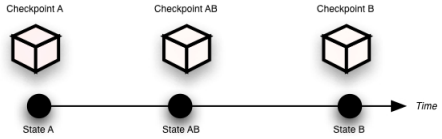

Figure 4: The retrospective creation of a known state.

the 100 what-if scenarios outlined in the previous example could be run with fewer wasted timesteps in the following way. We run from A to AB, call this AAB to see if the bifurcation has occurred. If it has we examine the section AAB in final detail and then run the trajectory ABB (AB to B). This is similar to the problem of determining roots of equations and it is of course possible to employ more sophisticated algorithms especially if we have some knowledge of the system, as illustrated in Figure 4.

By such methods we can locate the bifucation much more precisely on some segment. The task of retrospectively creating this checkpoint resolves to returning the simulation to the state that it was in when it originally transited State AB. This could be achieved by recording the state of the simulation at every time-step. However, this would impose too great an overhead. Instead, what is required is a lightweight approach to this problem that imposes minimal cost. A related approach is evident in the VisTrails system[2] which has adopted an interesting tree structure for visualising dataflows where nodes contain the operations that represent the "change actions applied to the parent to obtain the dataflow for the child node" [6]. This structure provides instructions for creating child dataflows from parent dataflows and so is of interest those trying to retrospectively create state in a lightweight manner. However, our approach is distinguishable from the VisTrails system as it provides the infrastructure by which the scientist can alter the checkpoint tree to make it a more efficient stucture for exploring changes in state. In the next section, we outline such an approach.

3.1 The Load, Replay & Save approach

The Load, Replay & Save (LRS) approach enables the scientist to retrospectively create a checkpoint of a previously transited state. In order to enjoy this additional functionality there is a necessary prerequisite, namely an accurate record of steered parameter changes. The record of steered parameter changes is necessary so that when the simulation is restarted any changes that were originally induced by steered parameter changes can be reapplied. The approach has the following steps:

1. **Load the checkpoint:** This step requires that the simulation is loaded with the checkpoint that precedes the bifurcation. This will be the starting point for the recreation of state.

2. **Restart the simulation:** At this stage the scientist must prepare the simulation to take a checkpoint at a specific point between State A and State B, namely State AB. State AB is the moment where the scientist wants to repeatedly restart the simulation from. In order to achieve this, the scientist will need to describe the exact timestep or, if that is not known, the parameter values that exist immediately prior to the bifurcation. Once the criterion for State AB has been declared the simulation can be restarted.

3. **Insertion of steered parameter changes:** As the simulation restarts, steered parameter changes are inserted into the simulation at the appropriate junctures. After each step in the simulation, the monitored and steered parameters are compared to checkpoint recreation criteria specified in step 2. If the conditions are met, Checkpoint AB is created, if not, the simulation increments to the next step. It is conceivable that the condition clause in Step 2 could be poorly formed and that on resumption of the simulation that the condition is never satisfied. Consequently, the simulation will also terminate if it reaches the parameter values defined in Checkpoint B.

4. **Creation of the checkpoint:** The final step is the creation of Checkpoint AB. This happens when the criteria defined in step 2 is evaluated as true.

4 A simulated example

We have developed a Java simulation for three reasons. Firstly, it in order to demonstrate the principles of the LRS approach. Secondly, using the Java simulation it is possible to enhance and refine the LRS approach through extended

[2] `http://www.vistrails.org/`

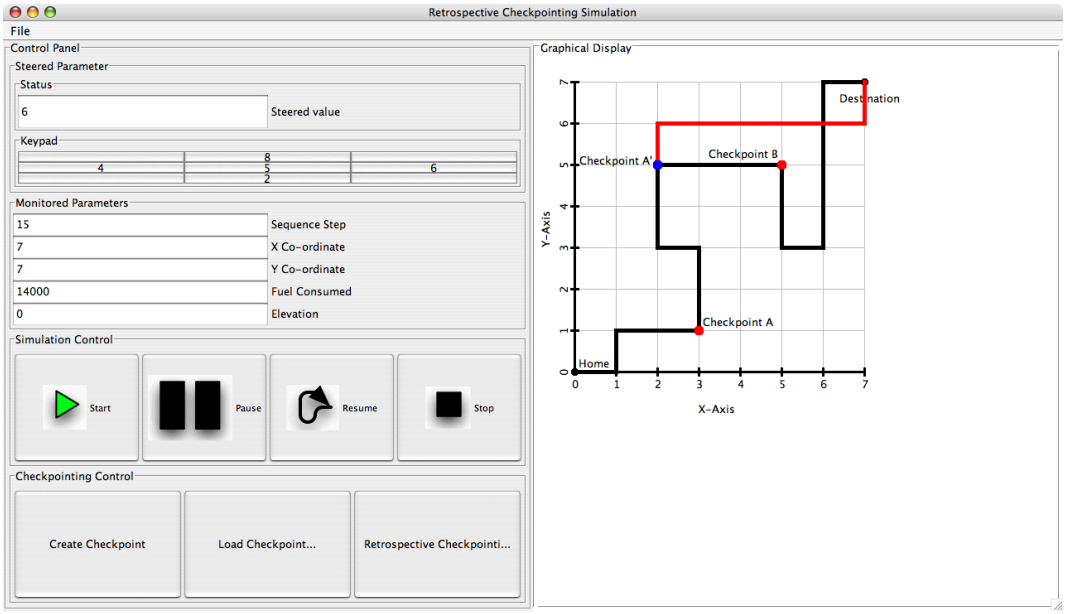

Figure 5: The simulation GUI showing controls and visualisation. The original route is shown as a black line, with checkpoints as red circles. The route taken from the retrospectively created checkpoint is shown as a red line.

use. Finally, and perhaps most importantly, the Java simulation will be used to demonstrate and quantify the efficiency improvements that retrospective checkpointing affords.

4.1 The basics

The simulation models a cross-town journey and attempts to discover the most *fuel efficient* route. The town is designed to an American-style Gridiron plan. At a basic level, the simulation models a vehicle moving over a two-dimensional surface. The simulation enables a driver to steer a vehicle from A to B. We describe Point A as *home* and Point B as *destination*. The size of the surface can be chosen by the user. The advantage of the Gridiron plan is that the roads are of uniform section lengths and intersect a regular intervals. This layout simplifies the movement that the simulation must model and limits the vehicle to orthogonal movements.

The simulation features computational steering, which allows the vehicle to be steered in real-time by keyboard or mouse inputs that relate to cardinal directions. Additionally, the simulation also provides checkpointing functionality that enables the user to save their progress, enabling the simulation to be resumed from *home* or from any state represented in a checkpoint. This level of functionality represents existing checkpointing capabilities.

Retrospective checkpointing functionality is also built into the simulation. This functionality enables state to be retrospectively created. In the example shown in Figure 5, the scientist may want to resume the simulation from a point that is not preserved in a checkpoint (e.g. where Y = 5 and X = 2) and retrospective checkpointing makes this possible. As previously mentioned, in order to support retrospective checkpointing, it is necessary to record changes to the steered parameter. Using the previous example Checkpoint AB is created by loading Checkpoint A and replaying the simulation until the state at State AB is reached. At this point, the simulation creates a new checkpoint called Checkpoint AB. Effectively, this new checkpoint has been retrospectively created. The advantage of this method is that only changes are journalled, not the full state of the simulation at each time-step; this keeps the overhead to a minimum. Retrospective checkpointing allows the scientist to try a different route to their destination from any point in the simulations history, rather than at the discrete points preserved in the existing checkpoint files. This is illustrated in Figure 5.

4.2 The usefulness of retrospective checkpointing

In order to demonstrate the benefit of retrospective checkpointing, a scenario has been created whereby the driver is attempting to find the

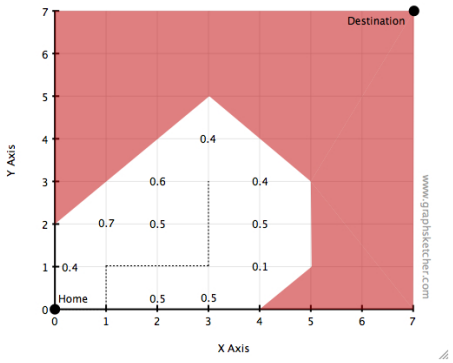

Figure 6: This diagram displays the elevation of the topography that the user has discovered.

most fuel efficient route across town. This task has been simplified so that only a single factor affects fuel consumption, the topography. In a real town, there would of course be other factors (e.g. congestion and traffic lights) but for this simulation the problem has been simplified. Every co-ordinate has an elevation property and the simulation has user-definable functions for determining the fuel consumption value. The basic rule is that each street requires a certain amount of fuel to drive along if the terrain is level. If the vehicle has to climb it uses more fuel and if the vehicle descends it uses less fuel. The object of the simulation is to *discover* the most fuel efficient route.

Although the topography is fixed at the beginning of the simulation, the values are unknown to the user. The topography is discovered as the vehicle travels through the terrain, revealing itself in only as far as the eye can see. This is approximated for our simulation by revealing the elevation of coordinates one move in every cardinal direction from the current location of the vehicle, as shown in Figure 6.

The simulation keeps a track of the fuel consumed by the vehicle and presents it to the user via the GUI. In addition, the user is given feedback on the terrain in each direction so that a steering choice can be made. In this scenario, it is banal that if the user drove into a hilly area of the topography, that they would want to return to a point when the terrain was more favourable. The retrospective checkpointing functionality in the simulation will allow the user to do just that, even if there was no existing checkpoint for that co-ordinate. By retrospectively creating a previously transited state, the user will be able to determine the most fuel efficient route more quickly than if the user had to restart the simulation from an existing checkpoint.

5 Critique of the LRS approach

Although work continues to add all the features to the Java simulation, there are some preliminary findings on the benefits and limitations of the LRS approach to retrospective checkpointing. In this section we discuss the assumptions we have made, the limitations of our approach and the benefits our approach will bring.

5.1 Assumptions

The LRS approach assumes the repeatability of simulations in a distributed Grid environment. This means that a simulation with the same starting state and the same steered parameter changes, run at different times, will return the same result. This assumption works on the basis that even though the physical hardware performing the simulation may differ, the outcome will be the same. It is clear that Grid technology has been designed to provide such an assurance and, whilst at a low-level there will be some differences due to differing hardware, firmware and software, the output of the simulation must be consistent. If this assumption proves to be false, the ability to retrospectively recreate an accurate checkpoint file will not be guaranteed.

There are essentially two accuracy issues implicit in advocating the LRS approach. The first surrounds the accurate recording of steered parameter changes. In order to accurately record the parameter values, the data-type of the object within the checkpoint will have to at least match the precision of the input steered parameter. If rounding occurs, it is assumed that this will have an impact on the simulation. The second accuracy issue relates to Step 3 and the extent to which one can be sure that steered parameter changes will be made at precisely the same moment that they originally occurred. The assumption is made that steered parameter changes can be guaranteed to be applied as per the original simulation.

The persistence of checkpoint data is essential to provide the ability to retrospectively create checkpoints. The LRS approach assumes that the checkpoint it references will persist long enough on the resource to support the retrospective recreation of a checkpoint of the scientists choosing. A possible way of achieving this will be by agreeing a data curation policy with Grid resource providers.

Finally, it is assumed that the scientist will be able to express, in terms of parameter values,

the point at which the bifurcation occurs. It is conceivable that this will not be the case if the characteristics of the bifurcation are particularly complex.

5.2 Factors supporting the LRS approach

As the situation presently exists, the computational scientist has, on the face of it, two options:

1. attempt to create another, more appropriate, checkpoint manually; or
2. accept an existing sub-optimal checkpoint.

The first option is essentially a manual version of the LRS approach. However, attempting to follow the LRS approach manually poses some significant obstacles if the state of the simulation prior to the bifurcation is to be accurately recreated. Firstly, the scientist would have to remember the steered parameter changes that were issued during the original running of the simulation. Once the simulation was restarted, the scientist would need to insert the steered parameter changes at the exact simulation step that they were originally applied. Finally, the scientist would need to precisely predict the imminent bifurcation. The difficulty of recreating accurately a previously transited state requires no elaboration. The obstacles that this option faces lead it to be discounted in all but the most simple of scenarios. Consequently, the choice given above is largely Hobson's choice—without the LRS approach, the computational scientist is forced to accept an existing sub-optimal checkpoint.

Given the scenario presented in Figure 3, the LRS approach clearly reduces the level of unnecessary computations that the simulation must conduct. The exact level of efficiency improvement will, of course, depend on the circumstances[3]. However, the scientist will still be able to choose to ignore LRS approach if the existing checkpoint is a close enough fit; in this sense, the LRS approach is entirely optional.

5.3 Limitations of the LRS approach

The focus of this paper has, so far, been on the provision of retrospective checkpointing functionality. However, the LRS approach is not a universal tonic and there will be times when the overhead associated with the generation of a retrospective checkpoint will outweigh the benefits of its existence. The cost trade-off is only one aspect. Perhaps of more importance is the issue of how the scientist interacts with this new functionality.

A specific problem relates to determining the correct checkpoint to load. Working out which checkpoint to load is non-trivial as the state of the simulation is not always immediately apparent from an inspection of the checkpoint tag and the exposed parameter values. A degree of trial and error in determining the checkpoint is anticipated.

Related to checkpoint determination is the extent to which the bifurcation can be expressed in terms of monitored or steered parameters. Expressing the bifurcation may be difficult if the parameter values are not precisely known. It is probable that the scientist would more naturally be able to express the bifurcation in terms of the visualisation of the simulation, rather than the express parameter values required by the LRS approach. Indeed, a more visual approach to checkpointing would doubtless provide a solution to the problems expressed above.

6 Conclusion and recommendations

It is evident that the current checkpointing functionality falls short of supporting the efficient use of Grid resources in certain circumstances. The LRS approach promises efficiency improvements over the status-quo by eliminating the multiple re-running of the previously transited states to support what-if scenarios. In essence, the LRS approach automates a process that the scientist would perform if only they could be accurate enough to do so.

However, before wholehearted endorsement of this approach, it is necessary to resolve some outstanding issues through further research. Firstly, a visual method of checkpoint and bifurcation selection needs to be established in order to make it easier and quicker for the scientist when interacting with the simulation. Secondly, more research is needed on how checkpoints can be made persistent, through data curation policy agreements with resource providers and identification of candidate checkpoints for deletion and retention. This is perhaps a key issue, as the persistence of this data across administrative domains seems difficult, if not impossible,

[3]The trade-off is between the cost of retrospective creation of a new checkpoint and the number of subsequent simulation iterations that take advantage of it.

to guarantee.

Assuming that work on the outstanding issues proceeds satisfactorily then we have little doubt that the retrospective checkpointing functionality, provided by the LRS approach, presents a model for improving the efficiency of the scientist engaged in computational research.

References

[1] Allen, G., Benger, W., Dramlitsch, T., Goodale, T., Hege, H., Lanfermann, G., A.Merzky, Radke, T., Seidel, E., and Shalf, J. Cactus tools for grid applications. *Cluster Computing 4* (2001), 179–188.

[2] Baillie, C., Michalakes, J., and Skålin, R. Regional weather modelling on parallel computers. *Parallel Computing 23* (1997), 2135–2142.

[3] Boghosian, B., and Coveney, P. Projects in scientific computing: Ketchup on the grid with joysticks. Pittsburgh Supercomputing Centre, 2004. `http://www.psc.edu/science/2004/teragyroid/ketchup_on_the_grid_with_joysticks.html`.

[4] Brodlie, K., Poon, A., Wright, H., Brankin, L., Banecki, G., and Gay, A. GRASPARC: A problem solving environment integrating computation and visualization. In *Visualization 93* (1993), IEEE Computer Society Press, pp. 102–109.

[5] Brooke, J. M., Coveney, P. V., Harting, J., Jha, S., Pickles, S. M., Pinning, R. L., and Porter, A. R. Computational steering in realitygrid. *Proceedings of the 2003 UK e-Science All Hands Meeting* (2003).

[6] Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., and Vo, H. T. VisTrails: Visualization meets data management. In *ACM SIGMOD* (Chicago, Illinois, USA, June 27–29 2006), pp. 745–747.

[7] Chin, J., Harting, J., Jha, S., Coveney, P., Porter, A., and Pickles, S. Steering in computational science: mesoscale modelling and simulation. *Contemporary Physics 44*, 5 (September–October 2003), 417–434.

[8] Coveney, P. V. Self-organization and complexity: a new age for theory, computation and experiment. *Phil. Trans. R. Soc. Lond. A 361* (2003), 1057–1079.

[9] Coveney, P. V., Jha, S., and Clarke, P. Spice: Simulated pore interactive computing experiment. *Proceedings of the 2005 ACM/IEEE conference on Supercomputing* (2005).

[10] González Sagredo, N., and Coveney, P. Cactus tools for grid applications. *Europhysics Letters 4* (2001), 795–801.

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

[12] Parker, S., and Johnson, C. SCIRun: a scientific programming environment for computational steering. In *Supercomputing '95* (New York, 1995), ACM Press, p. 52.

[13] Pickles, S., Blake, R., Boghosian, B., Brooke, J., Chin, J., Clarke, P., Coveney, P., González-Segredo, N., Haines, R., Harting, J., Harvey, M., Jones, M., Keown, M. M., Pinning, R., Porter, A., K.Roy, and Riding, M. The TeraGyroid experiment. In *Proceedings of GGF10* (2004).

[14] Pickles, S. M., Haines, R., Pinning, R. L., and Porter, A. R. Practical tools for computational steering. *Proceedings of the 2004 UK e-Science All Hands Meeting* (2004).

[15] Richardson, C. W., and Wright, D. A. *WGEN: A Model for Generating Daily Weather Variables.* Agricultural Research Service, US Department of Agriculture, 1984.

[16] Viceconti, M., and Clapworthy, G. The virtual physiological human: challenges and opportunities. *Biomedical Imaging: Macro to Nano* (2006), 812–815.

[17] Wozniak, S., and Smith, G. *iWoz - Computer Geek to Cult Icon: Getting to the Core of Apple's Inventor.* Headline Review, 2006.