Computational Steering Software Systems and Strategies

Steven G. Parker, David Beazley, and Christopher R. Johnson
Department of Computer Science
University of Utah, Salt Lake City, UT 84112.
E-Mail: sparker,beazley,crj@cs.utah.edu
Web: http://www.cs.utah.edu/~sci/

July 15, 1996

Contents

Abstract

Scientific visualization clearly plays a central role in the analysis of data generated by scientific simulations. Unfortunately, though visualization may in itself be more computationally intensive than the original simulation, it is often performed only as a mystical post-processing step after a large-scale batch job is run. An alternative to the usual scientific computing algorithm of performing modeling, computation, and visualization sequentially is to "close the loop" and interactively steer all phases of the application. In this paper, we discuss two approaches we have undertaken to interactively visualize and steer scientific applications.

The first of these approaches has been encapsulated in a software system called SCIRun. SCIRun is a shared memory based scientific programming environment that allows the interactive construction, debugging and steering of large-scale scientific computations. Using this "computational workbench," a scientist can design and modify simulations interactively via a dataflow programming model. SCIRun enables scientists to design and modify models and automatically change parameters and boundary conditions as well as the mesh discretization level needed for an accurate numerical solution. SCIRun is targeted for single workstation and SMP environments.

Many simulations, especially those running on MPP supercomputers, can run for hundreds of hours and generate gigabytes of data. In this case, scripting languages can be used quite successfully for steering applications. The second of these approaches, utilizes scripting language to add a steering capability to large-scale simulations running on both MPP systems and workstations. We also describe a tool called SWIG that we have used for building such applications. Within the context of the two steering systems, we describe large-scale applications in computational medicine and molecular dynamics.

Keywords

Computational steering, scientific computing, scientific visualization, modeling, simulation, scripting languages.

I. Visual Computing and Computational Steering

In recent years, the scientific computing community has experienced an explosive growth in both the size and the complexity of numeric computations possible. This growth is due, in large part, to both the rapid development of parallel machines and the rapid performance gains of RISC microprocessor technology. One of the significant benefits of this increased computing power is the ability to perform complex three-dimensional simulations. However, such simulations present new challenges for computational scientists. How does one effectively analyze and visualize complex 3D data? How does one solve the problems of working with very large datasets often consisting of tens to hundreds of gigabytes? How does one provide tools that address these computational problems while serving the needs of scientific users?

Scientific visualization clearly plays a central role in the analysis of data generated by scientific simulations. Unfortunately, though visualization may in itself be more computationally intensive than the original simulation, it is often performed only as a mystical post-processing step after a large-scale batch job is run. For this reason, errors invalidating the results of the entire simulation may be discovered only during post-processing. What is more, the decoupling of simulation and visualization presents serious scientific obstacles to the researcher. A visualization package may provide only a limited data analysis capability and may be poorly matched to the underlying physical models used in the simulation code. As a result, the researcher may expend significant effort trying to use a data analysis package only to walk away frustrated.

In 1987, the Visualization in Scientific Computing (ViSC) workshop reported [1]:

> Scientists not only want to analyze data that results from super-computations; they also want to interpret what is happening to the data during super-computations. Researchers want to *steer* calculations in close-to-real-time; they want to be able to change parameters, resolution or representation, and see the effects. They want to drive the scientific discovery process; they want to *interact* with their data.
>
> The most common mode of visualization today at national supercomputer centers is batch. *Batch processing* defines a sequential process: compute, generate images and plots, and then record on paper, videotape or film.
>
> *Interactive visual computing* is a process whereby scientists communicate with data by manipu-

lating its visual representation during processing. The more sophisticated process of *navigation* allows scientists to *steer*, or dynamically modify computations while they are occurring. These processes are invaluable tools for scientific discovery.

Although these thoughts were expressed close to ten years ago, they express a very simple idea that scientists want more interaction than is currently present in most simulation codes. While the scientific computing community is still trying to find better ways to address these needs, we feel that these problems encountered by computational scientists encompass a wider range of issues, including but not restricted to scientific visualization. In this paper, we describe our efforts to interactively control and steer complex scientific computations. Our efforts include a diverse range of techniques including, among others, the use of scripting languages, existing software, visual dataflow programming, and a sophisticated system designed exclusively for computational steering. We discuss several approaches we have taken to allow the steering of scientific applications. Two of these approaches have been singled out and pursued by our research group. The first of these approaches has been encapsulated in a program called SCIRun, and is targeted towards medium to large sized problems[1] that are run locally on a single high-end workstation or SMP supercomputer. The second of these approaches is targeted toward large-scale problems that are run on physically remote MPP systems. We will discuss these approaches and demonstrate their application to problems in computational medicine and molecular dynamics. We will compare the strengths and weaknesses of each approach and discuss ways to improve these models in the future.

II. Methods for Computational Steering

A. *Possible Approaches*

Computational steering requires integrating the modeling, computation, data analysis, visualization and data input components of a scientific simulation. The most critical aspect of any system is that it must allow for the efficient extraction of relevant scientific information, from simple x-y plots to sophisticated three-dimensional visualizations, as well as numerical quantities. Of course, this requires that the system be tightly coupled with the simulation code in order provide more information than would normally be available in a separate data analysis system.

A computational steering programming "model" is the software architecture used to integrate computational components in a manner that allows the efficient extraction of scientific information and permits changes to simulation parameters and data in a meaningful way. This new architecture often

[1] For the purposes of this discussion, we consider a "medium sized" problem to consist of hundreds of thousands of degrees of freedom while a "large sized" problem consists of millions of degrees of freedom

requires some modification of the original scientific code, but the extent and nature of the changes will depend on the model chosen. At one extreme, the scientific program will be completely rewritten and transformed to support steering. Less radical approaches may reuse pieces of the computation or use off-the-shelf visualization packages to simplify the construction of a steerable system [2], [3], [4].

Several approaches can be used to make scientific applications more interactive. Each approach has strengths and weaknesses. In many cases, a steering system will involve aspects of all of the approaches described here.

A.1 Program Instrumentation

One way to implement the steering of an existing scientific application is to make small modifications in the source to provide access points for the parameters and results. This process is called "instrumentation" and typically takes the form of subroutine calls inserted in the code wherever results become available or when new parameters can be used. These calls can transmit data to and from a separate visualization process. They might perform visualization tasks internally, or they might trigger a "leech thread," which siphons the data off while the computation continues concurrently.

This technique has the advantage of being minimally intrusive to an existing scientific code. However, it may provide only limited control over the computational process - the user gets to access only the parameters that have been instrumented. This technique also has implementation inefficiencies and complications, such as the overhead of data transmission if the data is being sent to a separate visualization process, the stalling of computation if visualization is done internally, and complicated synchronization if a leech thread is used. Furthermore, this approach does not work on many supercomputing systems using message-passing, as many such systems do not support multiple threads of execution because of synchronization problems and coding difficulty.

Interestingly enough, program instrumentation can often be accomplished using debugging tools. For example, the Prism debugger on the CM-5 allows users to visualize arrays and other complex structures within running programs [5]. Unfortunately, as a data-analysis tool, debuggers tend to make rather poor substitutes for a real data analysis system. A debugger may also create memory and performance difficulties within large-scale problems.

A.2 Directing Scientific Computation

An alternative approach to program instrumentation is to break a code up into various modules and allow the user to interact with them. Visualization and data analysis capabilities can easily be added as additional modules. To interact with the modules, one can use an interpreted scripting language such as Python or Tcl [6], [7]. This is exactly the model that has been successfully used in commercial packages such as IDL, MATLAB, or Mathematica. It is also currently being used in large physics applications at both Los Alamos and Lawrence Livermore National Laboratories [2], [8].

The advantage of this method is that it is possible to reuse almost all of the original scientific code. It is also portable and quite easy to implement on most systems since it avoids the problems of managing multiple threads and synchronization. The interpreted scripting language interface provides expert users with a fine degree of control over most, if not all, program parameters. This system can be also used for scripting long-running simulations, rapidly prototyping new features, and debugging.

While this approach is interactive, it is not highly interactive in a way that would allow a user to move the mouse and navigate through a dataset at high frame rates. However, systems based on this approach have been used quite successfully in a variety of scientific applications and data analysis packages.

A.3 Recasting Scientific Computations

If a scientist is fortunate enough to be designing a new model with computational steering in mind, there is more room for innovation. The first of the SCI group's computational steering systems, SCIRun [9], [10], has been designed for this scenario. It allows the scientist to construct a simulation using reusable computational components, connected within a visual programming environment. Each of these components, as well as the system as a whole, have been designed to integrate modeling, computation, and visualization, and to facilitate the interactive steering of all phases of the simulation.

In many instances, rewriting the scientific application may be the appropriate thing to do – if one is writing a new application, for example, or if there is just too much dust on the deck. However, there are times when starting over is not the wisest thing to do. At these times, it is important to allow reuse of significant portions of existing application components. SCIRun employs a somewhat flexible model, allowing existing scientific code to be encapsulated into SCIRun modules.

B. A Hierarchical Approach of Computational Steering

One of the primary goals of computational steering is to make scientific applications more interactive and flexible. Unfortunately, many traditional scientific codes fit into neither of these categories. Often, systems have been developed over a period of years or even decades. Adapting such applications to computational steering methods can be difficult. To further complicate matters, many steering systems are just as inflexible, forcing the user to adopt a particular methodology, code within a rigid hierarchy, or rely on a particular tool.

We feel that this rigidity simply is not acceptable given the long life-span of most scientific applications and the wide range of computational requirements found in these applications. Therefore, we

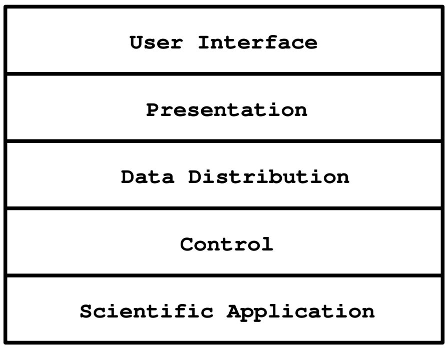

Fig. 1. Hierarchical Steering Model

are investigating a hierarchical approach to computational steering, as shown in Figure 1.

At the bottom of the hierarchy, we find the scientific computing application itself. This is the most important part of the hierarchy because this is where all of the physical models, numerical methods, boundary conditions, error estimation, data analysis, and other pieces of code are found. Given the difficulty of writing this code, it should be reusable in almost any application, independent of whether or not steering is involved (for example, it might be used to build a scientific programming library). Higher levels in the hierarchy logically represent different components that could be used in building a steering system. For example, one could build a steerable application using only a control language, or a highly sophisticated system involving high performance graphical output and real time input.

It is important to note that our hierarchical model does not imply a strict layering of components, but rather is a logical model for how we view computational steering systems. Each level in the hierarchy will generally involve components from all of the lower layers and will probably interact directly with the underlying simulation code in some capacity (and for performance). While keeping the simulation code intact is desirable, it is not a strict requirement, especially in cases where scientific applications have been recast into a steering model. However, even in such extreme cases, the hierarchical model still applies. We now describe each level of the hierarchy.

B.1 Control Mechanisms

In a steering system, a control mechanism is needed to drive the underlying simulation and data analysis code. In the case of program instrumentation, this might be a separate process that siphons off data for visualization and sends commands to the running process [4]. It might also be a scripting

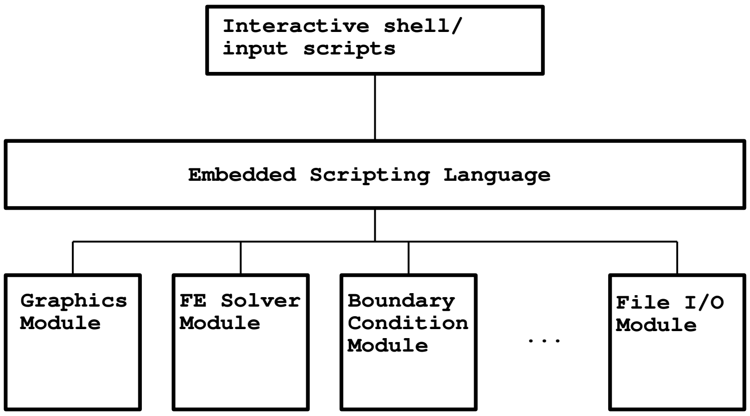

Fig. 2. A Programmable Scientific Application

language such as Python or Tcl. In either case, the control mechanism must be able to glue together and control different components and modules.

We have been using embedded scripting languages for controlling scientific applications. Scripting languages such as Python and Tcl are designed to interface easily with C/C++ code and work well for building modular and flexible applications. When used with a scientific application, it is possible to construct a programmable scientific application as shown in Figure 2.

Rather than thinking about a scientific application as a large monolithic system, we break up applications into a collection of modules. Each module may be a collection of C/C++ functions or a collection of procedures written in the scripting language itself. To users familiar with packages such as MATLAB or Mathematica, this approach is entirely familiar. A user can issue commands, run scripts, and extend the scientific application by writing procedures in the control languages.

Since most scripting languages now support dynamic loading, it is even possible to construct interesting scientific applications on-the-fly simply by assembling a collection of modules, issuing commands, and running a simulation. For example, the system shown in Figure 2 might be constructed in Python as follows :

```
>>> import graphics
>>> import fesolve
>>> import boundary
...
>>> import fileio
```

```
>>> ... start executing commands ...
```

Controls language have been used quite successfully in steering a variety of scientific applications [11]. While such systems usually do not provide graphical user interfaces, they work particularly well for large, long-running simulations, since these can be easily scripted yet run interactively when necessary. They are also attractive to experienced users who may appreciate having precise, low-level control over almost all aspects of their applications. It is also important to note that scripting languages can be used to build highly sophisticated graphical user interfaces that hide all of the low-level details.

B.2 The Data Distribution Layer

The data distribution layer addresses the problem of how data is handled by the steering system. In some cases, an entire problem may be run on a single machine. However, simulations may also be performed at remote sites such as national laboratories or supercomputing centers. Today, scientific simulations may easily generate tens to hundreds of gigabytes of data. As the problems of effectively dealing with this amount data have grown, some have even come to call such an excess of data a "data glut" [12]. While many steering systems have utilized high-speed networks between supercomputers and graphics workstations, we feel that much work is still to be done in this area.

Within the framework of distributed computational steering, the critical questions are, where do you put the data, and how do you analyze it?

- We could transfer all simulation data to the workstation where visualization and data analysis is performed. Unfortunately, this usually requires transmitting a very large amount of data between machines. Unless a high speed network is available, this may simply be infeasible. While such an approach allows for a great deal of interactivity in medium-sized problems, large datasets usually result in very poor interactivity. Of course this isn't so surprising, considering that the simulation data might have been generated on a machine tens to hundreds of times faster than a scientist's workstation.
- Rather than transferring all of the simulation data between machines, we could consider transmitting a subset of the data or some other intermediate representation. For example, graphics primitives might be transferred to the workstation rather than the raw data itself. Such an approach allows the user to efficiently manipulate the data for visualization purposes, but breaks down in systems with a high degree of geometric complexity. Quantifying and analyzing the results of a simulation may also be difficult if the workstation only has a subset of the simulation data.
- We can leave all of the simulation data on the remote system and attempt to perform data analysis

and visualization remotely. Under this scheme, a scientist's workstation might only receive image data. While this approach solves many of the problems mentioned earlier, it does not facilitate highly interactive viewing and manipulation of the data. For example, it might not be possible to simply drag the mouse and have an image rotate in real time.

There is no single approach that will work in all cases. Therefore, steering systems should be flexible enough to use aspects of all of these approaches. Certainly, there is a tradeoff between interactivity, network capacity, and desired functionality. There is probably much to be gained by developing clever data analysis and filtering techniques as well.

Data distribution remains an open problem in our computational steering research. We will return to the data distribution issue in the section on future work and limitations.

B.3 Presentation Layer

The presentation layer in the hierarchy describes the software systems used to present information to the end user of the steering system. This might include graphical user interface toolkits, graphics libraries, or software to control sophisticated output devices.

In our research, we have been primarily using Tk, an X11 toolkit that is integrated with Tcl (and also available for Python) to build graphical user interfaces. There are several reasons why this is a good idea. First, since Tk is integrated directly with popular scripting languages, it works nicely with applications that already use such languages as a control mechanism. Secondly, when performance is critical, Tk can be combined with high performance graphics libraries such as OpenGL or Open Inventor. Finally, Sun Microsystems has put considerable effort into making Tk a cross-platform system for developing graphical user interfaces on Unix, Macintosh, and Windows NT systems. Exploiting this fact will greatly simplify the process of developing steering systems as cross-platform support becomes more important. Recently, Sun released a Tcl/Tk plugin for the Netscape browser. Such a system may make it possible to use Tk as a mechanism for internet programming and distributed steering applications.

In other cases, it may be possible to use pre-existing software as a presentation mechanism. For example, a system may use public domain graphing libraries and image processing tools such as gnuplot and xv. Explorer, AVS, and other commercial systems have also been used successfully in steering systems [3].

B.4 User Interface Layer

Finally, at the top of the hierarchy is the mechanism by which the user interacts with the system. In other words, it is the means by which the user is able to interact with and extract scientifically meaningful information.

One important aspect of a computational steering system is the means by which it permits the user to specify changes to be made in the simulation. One issue is how to allow the user to specify changes in the computation. Various techniques may be used, ranging from simple text files to sophisticated scripting languages to graphical user interfaces to 3D widgets. Another less obvious issue is how to integrate changed data into the simulation in a scientifically meaningful fashion. In most coupled systems, it does not make sense to change one quantity without making corresponding changes in other quantities. For example, in a fluid dynamics system, it would not make sense to allow sudden changes in pressure without making corresponding changes in another quantity such as temperature in order to maintain balance under the ideal gas law.

Visual programming, 3D widgets, and graphical user interfaces all fall into this level. We return to this when we describe the SCIRun computational steering system that we have developed.

B.5 Putting it all together

Because computational scientists have a wide variety of needs, computational steering systems should be able to operate at different levels in order to serve different users and applications. For example, in a debugging or development mode, a researcher may want to use a highly interactive system that allows parameters and simulations to be run in almost real time. However, that researcher may later want to run large simulations requiring hundreds of CPU hours. In this case, it may be easier to use a control language and write batch scripts than to use an interactive environment.

To further complicate matters, a user may want to write extensions or use the system with their own applications. Unfortunately, doing so may require a detailed understanding of all of the middle layers of the hierarchy. Because they perceive the complexity of applying such systems to their own work, many potential users continue to ignore computational steering efforts. While there is no clear answer to this problem, we feel that this issue will need to be addressed in order for steering to be adopted into mainstream scientific computing efforts.

III. Computational Steering Efforts at Utah

A. Computational Steering with Scripting Languages

Many simulations, especially those running on parallel machines and supercomputers, can run for hundreds of hours and generate gigabytes of data. In this case, scripting languages can be used quite successfully for steering applications. There are several reasons for this:

- Scripting languages can be used to control batch simulations, a necessary component of most large-scale simulations.
- When using scripting languages interactively, a scientist can interactively issue commands to the language in order to perform data analysis and visualization (similar to using a package such as Mathematica or MATLAB).
- Scripting languages are memory efficient and can be used even when memory usage is critical.
- Having low-level access to the underlying data is important in many research applications in which much of the work is the development of new physical models, numerical methods, and data analysis techniques.

A.1 Automatic Module Generation with SWIG

To integrate a scripting language into a C/C++ application, it is necessary to write special "wrapper" functions that provide the necessary glue between the scripting language and C. Unfortunately, writing these wrapper functions is both tedious and error prone. To eliminate this problem, it is possible to use an automatic interface generation tool such as SWIG (Simplified Wrapper and Interface Generator) [13].

SWIG automatically generates the bindings between a variety of scripting languages and C/C++ programs using a specification file containing ANSI C/C++ declarations. A typical SWIG interface file (for the Los Alamos SPaSM molecular dynamics code) looks like the following :

```
%module crack
%{
#include "SPaSM.h"
%}

extern void     ic_crack(int lx, int ly, int lz, int lc,
                         double gapx, double gapy, double gapz,
                         double alpha, double cutoff);

/* Boundary conditions */

extern void     set_boundary_expand(void);
extern void     apply_strain(double ex, double ey, double ez);
extern void     set_initial_strain(double ex, double ey, double ez);
```

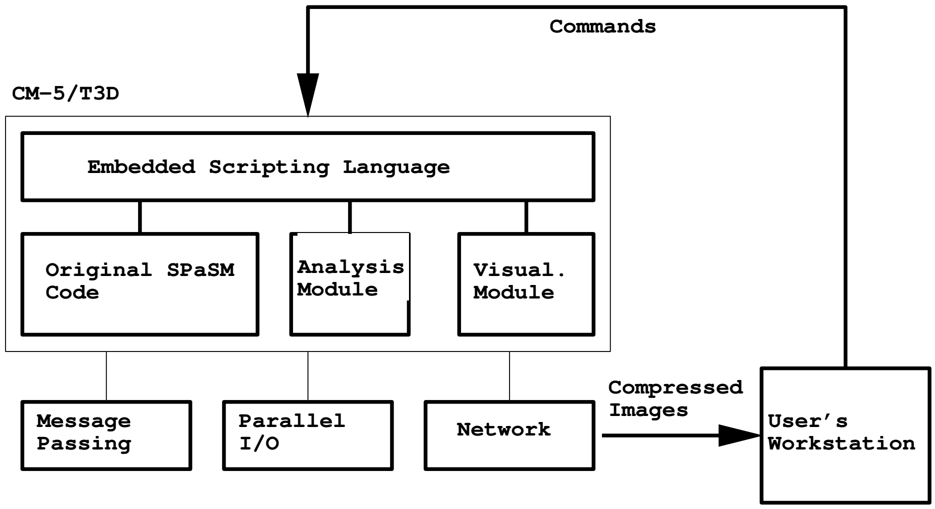

Fig. 3. Programmable version of SPaSM molecular dynamics code

```
extern void     set_strainrate(double exdot0, double eydot0, double ezdot0);
extern void     apply_strain_boundary(double ex, double ey, double ez);
```

At compile time, the interface file is automatically translated into C code that adds the specified functions to the interface. Since the process of integrating the C code with the scripting language is completely hidden, it appears to the user that the scripting language is interacting directly with the underlying C code. As a result, it is possible for the user to focus on scientific issues instead of worrying about systems programming issues.

A.2 Large-scale Molecular Dynamics

This approach has been used successfully in the SPaSM molecular dynamics code developed at Los Alamos National Laboratory [14], [15]. The SPaSM code was originally developed for use on the Connection Machine 5 and other massively parallel supercomputing systems. It is currently being used to study fracture, dislocations, and ductile-brittle transitions in materials. The code is written entirely in ANSI C and uses explicit message-passing for interprocessor communication. Prior to 1995, the SPaSM code ran exclusively in a batch processing mode and was controlled by a series of command line options. Typical production simulations ran for hundreds of hours, involve tens of millions of atoms, and produce tens of gigabytes of output files.

The SPaSM application now is controlled using a scripting language interface designed for massively parallel machines. The organization of the system is as shown in Figure 3. In the figure, the original molecular dynamics code is left mostly intact, but it is now combined with visualization and data

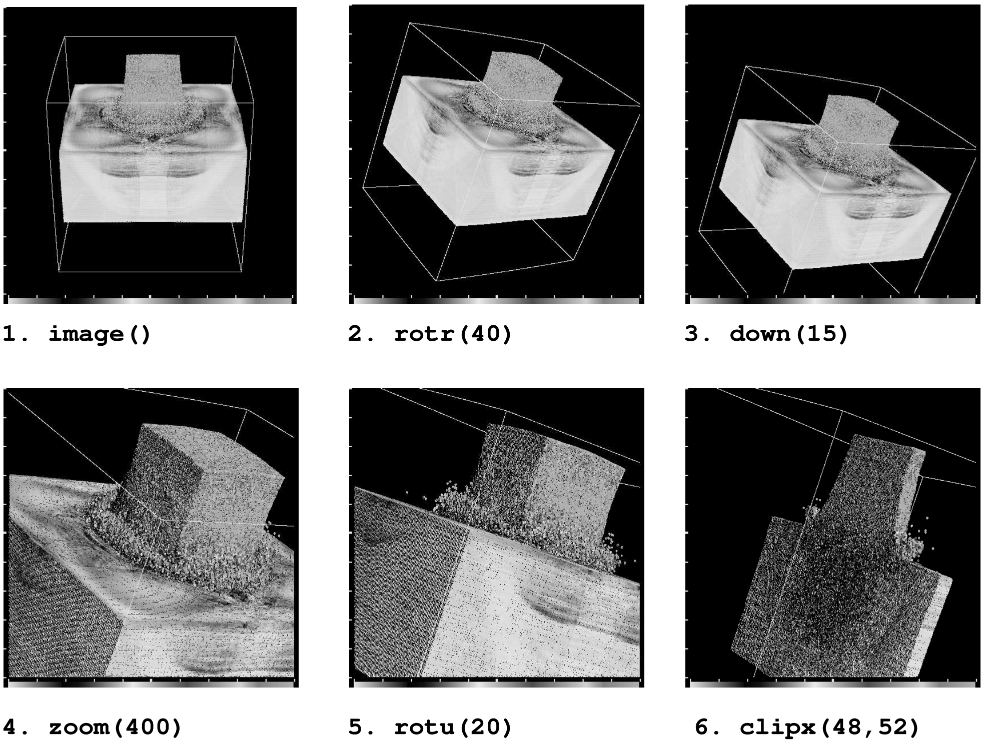

Fig. 4. Images generated by the SPaSM code. Each image was generated and displayed from 180 Mbytes of data in less than 15 seconds over a T1 internet connection.

analysis modules. Modules are added to the system using SWIG. The scripting language interface is primarily used for controlling batch jobs, but it can also be used interactively during post processing, setting up problems, and debugging. Due to the large dataset sizes, all visualization and data analysis is performed directly on the parallel machine. As output, GIF images are sent across the network to an image server running on the researchers workstation where the user can watch a simulation progress in real time. Alternatively, images can be dumped directly to disk for later examination.

When used interactively, the SPaSM system may be used as shown in Figure 4. The data is from an impact simulation involving 11 million atoms. Each dataset is approximately 180 Mbytes, but each image requires only about 15 seconds to generate and display remotely over a standard T1 internet connection between the University of Utah and Los Alamos National Laboratory.

While it is not currently possible to interact with the system using a GUI, it is possible to generate images of as many as 150 million particles in less than 30 seconds. In contrast, it would take several days to download that data to a graphics workstation and work with it locally. The fine degree of control provided by the scripting language interface also works well in practice, since the users of this system are constantly making code modifications and adding new features. A more detailed description of this system will be published elsewhere [11].

B. The SCIRun Computational Steering System

Much of the computational steering research at Utah has focused on SCIRun, an interactive environment for creating and steering scientific applications [9], [10], [16]. SCIRun is a scientific programming environment that allows the interactive construction and steering of large-scale scientific computations. A scientific application is constructed by connecting computational elements (modules) to form a program (network). This program may contain several computational elements as well as several visualization elements, all of which work together in orchestrating a solution to a scientific problem. Geometric inputs and computational parameters may be changed interactively, and the results of these changes provide immediate feedback to the investigator.

One particular application has driven the study of computational steering as an everyday research tool - modeling electrical conduction in the human thorax for the study of implantable defibrillator device designs. An implantable defibrillator is a small electrical device which is surgically implanted in the human chest, and discharges electrical shocks to restart the heart when it undergoes a heart attack. However, there are many variables in the design of such a device, and optimal operation will be achieved through the correct combination of electrode placement, size, shape and numbers. This large design space cannot be searched experimentally, but computational models can allow an engineer to test a wide range of possibilities. Results from such a simulation are shown in figure 6. In order to accurately model such a system, a highly detailed computational model must be used, and these models can take hours of CPU time to compute the results.

Using SCIRun, an engineer could interactively steer such a simulation. An engineer can try an electrode configuration and start the simulation. As the simulation progresses, the engineer can view the intermediate results. Long before the system completes a detailed solution, the engineer might determine that the configuration is not acceptable, and would try a new configuration and restart the simulation. Instead of throwing everything away and starting over, SCIRun uses previous results to jump-start the new simulation. In scenarios where only small changes are made to input parameters, a significant CPU savings can be achieved.

There are many engineering design problems which would benefit from such a system, ranging from the biomedical problems discussed here, to traditional mechanical design, and even CFD aerodynamics design. These engineers desire a tool where they can easily experiment with new geometric configurations, and can also modify computational models and numerical algorithms to achieve more accurate results, or to compute the solution more efficiently. As an integrated visualization environment com-

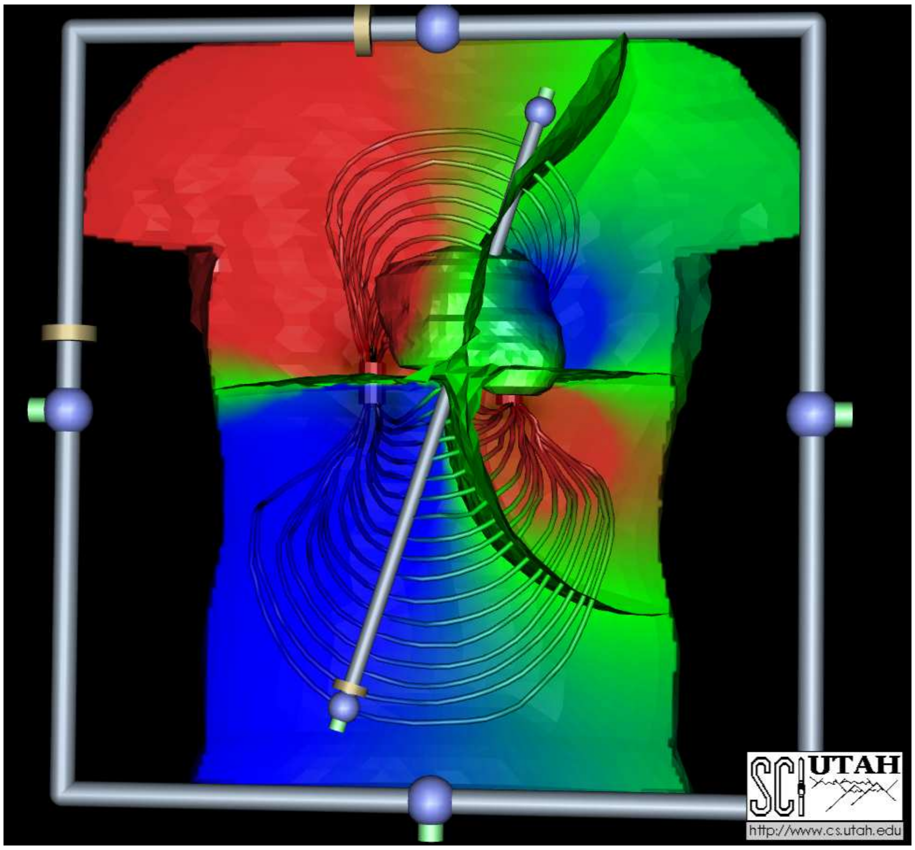

Fig. 5. 3D adaptive finite element solution of an internal defibrillation simulation. A user can interactively investigate the isovoltage surfaces, the lines of electric current flow, as well as change the position and shape of the electrode source.

bined with large-scale computational capabilities, SCIRun is a powerful tool in engineering design.

A SCIRun dataflow network, which implements an adaptive finite element solution for the inhomogeneous Laplace's, equation is shown in Figure 7. This example illustrates the power of SCIRun to build a complex scientific simulation using a handful of core steerable computational components. It uses a dataflow feedback loop to implement adaptive mesh refinement. and allows a scientist to interactively vary mesh adaptation parameters, solution methods, and boundary conditions, and to simultaneously visualize the resulting solutions.

However, SCIRun is currently limited to execution on a single workstation or SMP, which restricts the size of the problems that can be attempted. Future additions to SCIRun will extend it to a distributed cluster of workstations or large multiprocessor supercomputers.

IV. Future Directions of Computational Steering

Since the 1987 ViSC report, computational steering has created an evolutionary acceptance within, rather than a revolutionary change to, the scientific computing process. This trend is likely to continue,

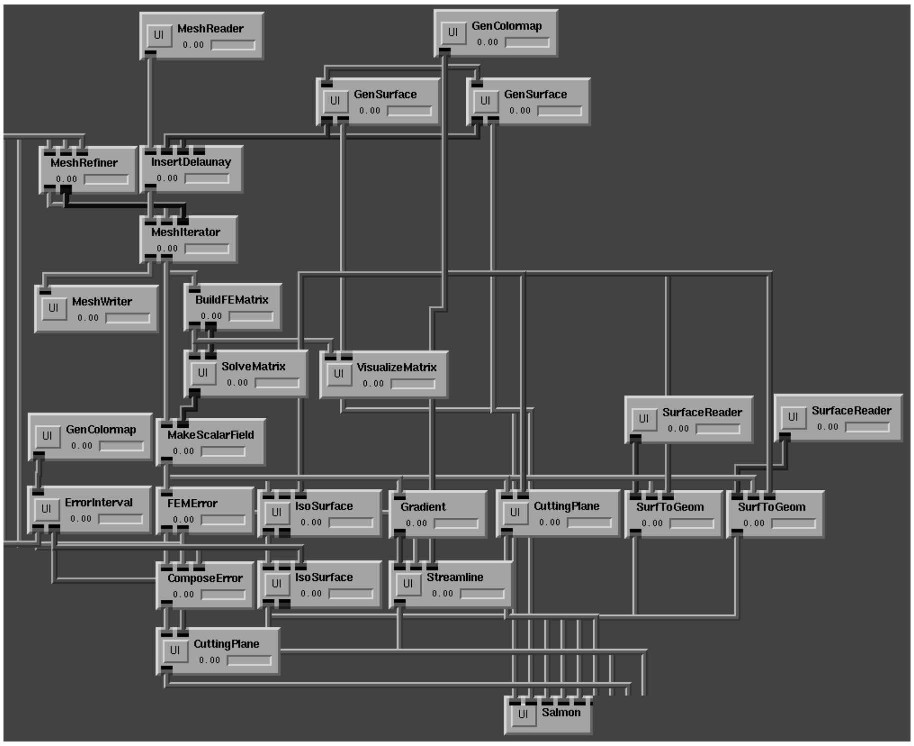

Fig. 6. Adaptive finite element dataflow program.

but some factors suggest that the evolution may accelerate:

- The melding of high-end graphics workstations and supercomputers will allow many more scientists to perform computations and visualization using the same hardware. Instead of transferring gigabytes of data to a visualization workstation, the visualization will be performed directly on the supercomputer. During the development phases of a scientific model, the scientist may be able to completely avoid writing data to the disk. Expensive disk writes will only be necessary for production runs.
- Simulation is playing an increasingly important role in today's scientific and engineering world. Increased safety responsibilities, heightened environmental awareness, and cheaper CPU cycles have all increased the motivation for many engineers to simulate everything.
- Visualization systems are becoming more accessible to average scientists. A few years ago, most scientists didn't even have ready access to color workstations, but now they are commonplace. Similar changes are in the pipe for 3D graphics engines, which will make moderately fast graphics machines available at a reasonable cost.

However, there are barriers to this progress:

- Supercomputers are currently set up for batch mode processing, not for interactive use. A computational steering system will break many of the assumptions that are made in these batch mode systems. An important research project would be to investigate ways of efficiently sharing supercomputer resources in an interactive computational steering world.

- Since accounting systems at supercomputer sites focus on counting CPU cycles, performing visualization and rendering tasks using these cycles can be extremely expensive. These costs are highly relevant to the large-scale problems run on large supercomputers, but are not relevant to the scientist doing medium-sized problems on his desktop superworkstation. There is also room for the investigation of tightly coupled "visualization engines," smaller machines placed at the supercomputer center site that can be used to offload visualization tasks from the primary compute machine.

Future research will need to address these barriers, and the policies and practices of supercomputer sites will need to be flexible in order to make interactive supercomputing feasible.

A. Strengths and Weaknesses of Computational Steering

Computational Steering has been very successful in augmenting the development process - it is very easy to detect bugs in your code using the visualizations, and it is often easy to postulate the source of a bug when seeing the erroneous results.

For example, in the SPaSM molecular dynamics code, pictures can help detect problems with setting up boundary conditions and initial conditions. In incorrect cases, it possible to visually detect sound waves and other disturbances as they are emitted from boundary regions. Instead of finding out that this has occurred *after* a production simulation, it is often possible to observe this phenomenon by running only a few hundred time steps and watching the evolution of the system in real time.

Similar methods have been used in SCIRun for boosting confidence in the accuracy of a simulation. For example, when modeling a system governed by an isotropic Laplace's or Poisson's equation, is is easy to visually verify that the electrical current lines are perpendicular to the isosurfaces, that the isosurfaces are perpendicular to boundaries, and that the streamlines do not leave a closed boundary. While these visual inspections do not prove that a simulation is correct, they can provide a quick "believability" check when code is modified.

Unfortunately, the scientific community has been slow to adopt the method of computational steering in everyday research applications. This may be due to a variety of reasons :

- Many scientists perceive computational steering systems as being too complicated to use and modify. While this may or may not be the case, in order to be successful, we feel that steering systems will need to emphasize simplicity of use while providing powerful scientific tools. Most scientists have already put a considerable effort into the development of simulations. While there may be little incentive to rewrite these systems, we can only hope that steering is seen as a desirable capability that can be incorporated into new applications or applied to existing systems with minimal effort.

- Computational steering has not been cost effective. Many steering systems rely on high-end graphics workstations, yet many scientists do not have the resources to buy such equipment. This may only be a temporary problem as the price of high-performance of graphics hardware continues to drop, but in the future it will be important for steering systems to target systems that are likely to be found on a researcher's desk instead of those found in supercomputing centers.
- Computational steering systems must represent a balance between interactivity, batch processing, cost, and the ability to extract scientifically useful information. For example, a system that provides 30 frames per second of output to a user, yet provides no mechanism for long running simulations may be of only limited usefulness. Likewise, a system with good quantitative features, but poor interactivity may also be inadequate within a steering context. In any case, we feel that there will be little incentive for a researcher to start using a computational steering system unless it is balanced and it supports a variety of operational modes.
- Researchers fear that computational steering systems will consume precious resources, such as CPU time, memory and disk space. Where this is a concern, steering systems should be as trim and efficient as possible, and should be able to "get out of the way" for long/large runs where resources are limited. It is also necessary for a system to be worth their cost - they should save a researcher more in save time and CPU cycles than they consume. While it is difficult to measure this effect directly, it will be reflected in the acceptance of computational steering as a ubiquitous tool in the computational scientist's workbench.

B. Future Directions in Computational Steering

Computational steering will succeed only if steering systems can be useful to scientists and engineers. Systems should be modular and easy to extend with existing code. Users of such systems should be able to add new capabilities easily without being overwhelmed in systems programming issues. Steering systems should be adaptable to hardware ranging from the largest of supercomputing systems to low-end workstations and PCs. Finally, steering systems must be demonstratably usable in scientific research. Working prototypes are a start, but until scientists and engineers start using such systems regularly, the use of steering systems will remain evolutionary in nature.

At Utah, we have developed two different approaches for computational steering. On the one hand, we have a low-level approach that utilizes scripting languages and works well with batch processing and large scale simulations while providing a fine degree of control to users. The system also works on inexpensive UNIX workstations. Unfortunately it's not very interactive. On the other hand, the SCIRun system provides extensive support for interactively manipulating complex datasets, performing visualization, and making changes to program parameters. Unfortunately, SCIRun does not currently work well with very large datasets or long-running simulations, and it requires graphics workstations for good interactivity.

Our two approaches provide a unique opportunity for exploring innovative steering systems for a variety of applications. Much of our future work will explore the distribution of data and the combination of command driven techniques with the dataflow approach used in SCIRun. We hope to investigate the use of systems such as ILU [17] and CORBA in order to build distributed steering systems involving multiple platforms and languages. Ultimately, we hope that this can lead to the development of steering systems that are highly interactive, easy to extend and modify, work well with large amounts of data, and which can be used on ordinary workstations, high end servers, and supercomputing systems.

As steering systems become more common, issues related to data management will become critical. It easy for a researcher to generate hundreds or even thousands of images, datafiles, and results. Organizing that data is a significant problem. Thus, much work will need to be done to effectively organize that data. Scientific database research may have much to offer in this area.

V. Acknowledgments

This research was supported in part by awards from the NIH, NSF, and DOE. The SPaSM molecular dynamics code was developed at Los Alamos National Laboratory under the auspices of the Department of Energy with Peter Lomdahl, Brad Holian, Shujia Zhou, Tim Germann, and Niels Jensen of the Theoretical Division. We also acknowledge facilities from the Advanced Computing Laboratory at Los Alamos National Laboratory and the NSF STC for Computer Graphics and Scientific Visualization. The authors would like to thank K. Coles, J.D. Brederson, and Y. Livnat for their helpful comments and suggestions.

References

[1] T. De Fanti et al. Special issue on visualization in scientific computing. *Computer Graphics*, 21(6), November 1987.

[2] T-Y.B. Yang, P.F. Dubois, and Z.C. Motteler. Building a programmable interface for physics codes using numeric Python. *Proc. of the 4th International Python Conference*, 1996.

[3] J.A. Kohl and P.M. Papadopoulos. A library for visualization and steering of distributed simulations using PVM and AVS. *High Performance Computing Symposium*, Spring, 1995.

[4] Y. Jean, T. Kindler, W. Ribarsky, W. Gu, G. Eisenhauer, K. Schwan, and F. Alyea. An integrated approach for steering, visualization, and analysis of atmospheric simulations. In *IEEE Visualization '95*, pages 411–418, 1992.

[5] Thinking Machines Corp. *Prism Users Guide*. 1994.

[6] M. Lutz. *Programming Python*. O'Reilly and Associates, Sebastopol, CA, 1996.

[7] J. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, New York, 1994.

[8] P. Dubois. Making applications programmable. *Computers in Physics*, 8(1):70, 1994.

[9] C.R. Johnson and S.G. Parker. A computational steering model for problems in medicine. In *Supercomputing '94*, pages 540–549. IEEE Press, 1994.

[10] C.R. Johnson and S.G. Parker. Applications in computational medicine using SCIRun: A computational steering programming environment. In *Supercomputer '95*, pages 2–19. Springer-Verlag, 1995.

[11] D.M. Beazley and P.S. Lomdahl. Lightweight computational steering of very large-scale molecular dynamics simulations. *Supercomputing '96*, 1996 (to appear).

[12] S. Bryson. The data glut revisited. *Computers in Physics*, 9(5):525–530, 1995.

[13] D.M. Beazley. SWIG : An easy to use tool for integrating scripting languages with C and C++. *Proceedings of the 4th USENIX Tcl/Tk workshop*, 1996 (to appear).

[14] D.M. Beazley and P.S. Lomdahl. Message-passing multi-cell molecular dynamics on the Connection Machine 5. *Parallel Comp.*, 20:173–195, 1994.

[15] P.S. Lomdahl, P. Tamayo, Gronbech-Jensen, and D.M. Beazley. 50 gflops molecular dynamics on the Connection Machine 5. In *Proceedings of Supercomputing '93*, pages 520–527. IEEE Press, 1993.

[16] Parker. S.G. and C.R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Supercomputing '95*. IEEE Press, 1995.

[17] B. Janssen and M. Spreitzer. ILU : Inter-language unification via object modules. *OOPSLA 94 Workshop on Multi-Language Object Models*, 1994.

Steven G. Parker is currently pursuing his Ph.D. in the Department of Computer Science at the University of Utah. In 1994, he received a DOE Computational Science Graduate Fellowship. His research interests include scientific visualization, computer graphics, user interaction and scientific computing. Recent work has focused on development of the SCIRun computational steering software system.

David Beazley has worked in the Condensed Matter and Statistical Physics group of the Theoretical Division at Los Alamos National Laboratory since 1991. His research interests have included massively parallel computing, molecular dynamics modeling of materials, portability and performance issues of large-scale parallel computing, and numerical methods. Currently, he is pursuing a Ph.D. in computer science at the University of Utah, where he is working on cost effective methods for analyzing and managing very large scientific datasets, software development tools for scientific computing, and scientific databases. He is also the person responsible for SWIG.

Christopher Johnson received his Ph.D. from the University of Utah in 1989. He is an Associate Professor of Computer Science at the University of Utah. His research interests are in the area of scientific computing. Particular interests include inverse and imaging problems, adaptive methods for partial differential equations, large-scale computational problems in medicine, computational steering, and scientific visualization. Professor Johnson was awarded a FIRST Award from the NIH in 1992, a National Young Investigator (NYI) Award from the NSF in 1994, and the Presidential Faculty Fellow (PFF) Award in 1995. He directs the Center for Scientific Computing and Imaging at the University of Utah.