# An Integrated Problem Solving Environment: The SCIRun Computational Steering System

Steven G. Parker, Michelle Miller, Charles D. Hansen and Christopher R. Johnson
Department of Computer Science, University of Utah, Salt Lake City, UT 84112
{sparker,mmiller,hansen,crj}@cs.utah.edu, http://www.cs.utah.edu/~sci

**Abstract**

*SCIRun is a scientific programming environment that allows the interactive construction, debugging, and steering of large-scale scientific computations. We review related systems and introduce a taxonomy that explores different computational steering solutions. Considering these approaches, we discuss why a tightly integrated problem solving environment, such as SCIRun, simplifies the design and debugging phases of computational science applications and how such an environment aids in the scientific discovery process.*

## I. Introduction

Since the introduction of computers, scientists and engineers have attempted to harness their power to simulate complex physical phenomena. Today, the computer is an almost universal tool used in a wide range of scientific and engineering domains.

Computational science and engineering is the field that has grown out of the widespread use of computers to numerically simulate the physical phenomena associated with many problems in science and engineering. In a typical scenario, a computational scientist follows this algorithm:

**Construct a model of the physical problem domain.** Specify the shape of the problem domain, as well as other physical properties, such as electrical conductivity, density or viscosity. Simple problems may have relatively simple models, such as cubes, spheres or other simple geometries. However, current trends typically require the use of "real" life models that accurately portray a related physical problem domain. For example, computational medicine problems typically addressed by the Utah Scientific Computing and Imaging (SCI) group involve creating a detailed model of the human anatomy which describes the shape and electrical conductivities for the bones, muscles and organs in a human torso [1]. Modeling may also include the specification of initial conditions for the simulation, such as the current weather conditions for a weather simulation.

**Apply boundary conditions.** Boundary conditions are the forces that drive a particular problem. Typical boundary conditions may include the velocity of wind at the input of a wind tunnel, the electrical sources for an electrical problem, or boundary temperatures for a heat conduction problem. Conditions are defined on a boundary that couple with the governing equations to define the behavior of the system at these boundaries. Parameters to these equations may also be specified in conjunction with other model parameters.

**Develop a numerical approximation to the governing equations.** Governing equations are a set of partial differential equations that define the behavior of the problem. Since the computer cannot operate on these equations directly, the equations are discretized using methods such as Finite Difference, Finite Element, Finite Volume, Boundary Element methods.

**Compute.** Once the data has been specified, the computer is used to solve this numerical approximation. This typically involves solving a linear or non-linear system of algebraic equations. For realistic models, these systems of equations can be extremely large, incorporating thousands to millions of unknowns.

**Validate the results.** Once the solution has been found, the scientist must determine if the results are correct. Validation methods include computing known problem invariants (a form of "checksum"), comparing results with experimental data, comparing results with those of simple problems with analytical solution, and determining that the answer is plausible according to the scientists' expertise on the problem.

**Understand the results.** Early scientists printed out stacks of numbers on continuous sheet line printers and stared at them for hours. As computers grew more powerful, scientists were able to perform more complex simulations. Fortunately, these more powerful computers are able to present information in a more meaningful way using Scientific Visualization.

Over the years, scientific computing has grown into a widely accepted method of scientific investigation. Scientists are continuously trying to perform more accurate simulations, to create more realistic physical models, and to obtain solutions in less time with less work. Many scientists are also applying these techniques to new problem domains and are using them to solve new practical problems.

### A. Computational Steering

Currently, organizing, running and visualizing a new large-scale simulation still requires hours, days or weeks of a researcher's time. Data I/O and conversion time further complicates and slows the process. Even for experienced scientists who may employ scripts and conversion programs to aid them in the task, the process is anything but streamlined. As noted at the NSF sponsored workshop on Health Care and High Performance Computing in 1994, scientists and engineers want a system in which all these computational components are linked. In other words, they wish to "close the loop," so that all aspects of the modeling and simulation process can be controlled graphically within the context of a single application program. In areas such as healthcare, researchers cannot afford the predominant batch-mode approach since their applications are time critical in nature.

In 1987, the Visualization in Scientific Computing (ViSC) workshop reported [2]: "Scientists... want to drive the scientific discovery process; they want to *interact* with their data. *Interactive visual computing* is a process whereby scientists communicate with data by manipulating 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 recorded ten years ago, they express a very simple, still current idea: that scientists want more interaction than is permitted by most simulation codes. Computational steering has been defined as "the capacity to control the execution of long-running, resource-intensive programs" [3]. In computational science, we apply this concept to link visualization with computation and geometric design to interactively explore (steer) a simulation in time and/or space. As the application is developed, a scientist can leverage the steering and visualization to assist in the debugging process as well as modify the computational aspects based upon performance feedback. As knowledge is gained, a scientist can change the input conditions, algorithms, or other parameters of the simulation.

Implementation of a computational steering environment requires a successful integration of the many aspects of scientific computing, including performance analysis, geometric modeling, numerical analysis, and scientific visualization. These requirements need to be effectively coordinated within an efficient computing environment (which, for large-scale problems, means dealing with the subtleties of various high-performance architectures).

Recently, several tools and environments for computational steering have been developed. These range from tools that modify performance characteristics of running applications, either by automated means or by user interaction, to tools that modify the underlying computational application, thereby allowing application steering of the computational process. Our view is that a Problem Solving Environment (PSE) that encompasses all of these characteristics, from algorithm development through performance tuning to application steering, for scientific exploration and visualization and provides a rich environment for accomplishing computational science.

In the remainder of this paper, we first describe the application of a system we have developed, SCIRun, to the domain of computational field problems. Next, we briefly describe the SCIRun software architecture. We review related work and introduce a taxonomy that explores different computational steering solutions. We then present our thoughts on why a Problem Solving Environment such as SCIRun is crucial to computational science and engineering.

## II. Computational Field Problems and SCIRun

SCIRun is a scientific programming environment that allows the interactive construction, debugging, and steering of large-scale scientific computations. The primary purpose of SCIRun is to enable the user to interactively control scientific simulations while the computation is in progress. This control allows the user, for example, to vary boundary conditions, model geometries, and/or various computational parameters during simulation. Currently, many debugging systems provide this capability in a low-level form. SCIRun, on the other hand, provides high-level control over parameters in an efficient and intuitive way through graphical user interfaces and scientific visualization [4], [5]. These methods permit the scientist or engineer to "close the loop" and use the visualization to steer phases of the computation.

The ability to steer a large scale simulation provides many advantages to the scientific programmer. As changes in parameters become more instantaneous, the cause-effect relationships within the simulation become more evident, allowing the scientist to develop more intuition about the effect of problem parameters, to detect program bugs, to develop insight into the operation of an algorithm, or to deepen an understanding of the physics of the problem(s) being studied.

Initially, we designed SCIRun to solve specific problems in Computational Medicine [1], [6], but we have made extensive efforts to make SCIRun applicable in other computational science and engineering problem domains. In addressing specific problems, we found that there were a wide range of disparate demands placed on a steerable problem solving environment such as the SCIRun system. We now provide a more detailed discussion of solving computational field problems.

### A. Geometric Modeling

In most computational engineering and science applications, significant geometric modeling must take place prior to simulation and visualization. Modeling efforts usually involve geometrical construction of a physical domain, in which a continuous structure must be discretized and adequately rendered into discrete spatial elements.

Construction of the geometric model is often one of the most time consuming aspects of modeling and simulation. For each new configuration, a new model must be assembled. Once a model is up and running simulations, a researcher must wait through an entire simulation before making changes to the geometry or before learning if the changes already enacted have been effective. Because making such changes and recomputing the effects of those changes is very time consuming, researchers are often restricted in the number of options they can effectively test.

In the SCIRun computational steering system, a goal is to change geometric features of the model or the spatial discretization of the solution domain interactively. Ideally, the user receives some degree of feedback on the calculation almost immediately, and is allowed to change input boundary conditions, such as spatial location and magnitude of a source, or the timestep with which the calculation proceeds. These changes automatically trigger the computational and visualization phases of the problem. Such an environment allows more immediate access to simulation results, significantly reducing the time spent in making simulation and modeling design changes.

### B. Numerical Analysis

A variety of techniques are used to numerically approximate the partial differential equations (PDEs) that govern most computational field problems. We discuss the finite element (FE) method here, although most of the concepts apply to finite difference, boundary element, and multigrid methods. Application of the FE method yields a linear system, $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is the so-called "stiffness matrix" and can vary from hundreds of thousands to millions of degrees of freedom.

For solving this system, the scientist can choose from a variety of direct and iterative solution methods, as well as from different preconditioners. A user interface is provided to change tolerances, maximum iteration counts, and other numerical parameters. During the solution process, SCIRun provides feedback on several numerical and performance parameters, such as residual error, iteration count, MFLOPS, etc. The scientist can also interactively decide upon the level of accuracy used for a given simulation based upon *a priori* design criteria. Upon initiating the simulation, the scientist views the initial results and is presented with a visual representation of the computation's effectiveness (based on various quantitative measures, such as the error per element of the finite element analysis). Then, the scientist decides if (s)he would like to continue the computation using a more (or less) refined level of discretization or restart the computation with different input conditions.

### C. Scientific Visualization

Certainly, effective interpretation of computer simulations depends upon the visualization of the data. Traditionally, visualization has been entirely separate from the computation phase. Computations were stored off to disk and/or piped into a separate visualization software package once all computations are completed. Furthermore, many scientists relied on current "off the shelf" visualization packages that are not well suited for use with large engineering datasets (at least not in an interactive fashion). Within SCIRun, visualization is an integral part of the computational and geometrical modeling phases The user is able to visualize and explore intermediate results while the calculations continue to progress. Refined datasets are automatically substituted for the less accurate ones as they are completed.

SCIRun brings together a large collection of algorithms for realizing the components of a scientific computing environment outlined above. Connecting these algorithms in an efficient manner into a flexible environment contributes to the computational steering goal. Although creating new modules for the system is ongoing work, current efforts have concentrated on building an integrated and interactive environment for solving large-scale computational field problems.

## III. SCIRun - A Computational Steering Environment

SCIRun is a problem solving environment in which large scale computer simulations can be composed, executed, controlled and tuned interactively. Composing the simulation is accomplished via a visual programming interface to a dataflow network. Software systems such as AVS from Advanced Visualization Systems Inc.[7], Iris Explorer from NAG, and Visualization Data Explorer from IBM [8] have made this archetype popular for scientific visualization [9]. Our work has extended this paradigm into the realm of scientific computation and steering.

To execute the program, one specifies parameters with a graphical user interface rather than with the traditional text-based datafile. Controlling a simulation involves steering the simulation interactively as it progresses. In SCIRun, the typical components of the computational paradigm – geometric modeling, numerical analysis, and scientific visualization – are integrated into a visual programming environment that provides the researcher with the ability to interactively steer any one phase of the process and to see the effects propagate throughout the system automatically.

As an example of the SCIRun system interface, see Figure 1. A graphical representation of the dataflow network is shown in the lower right. The boxes represent computational algorithms (modules), while lines represent data connections between the modules. Each module may have a separate user interface, such as the matrix solver interface at the left, that allows the user to control various parameters. An interactive 3D viewer that combines visualization output and data probes is found at the top.

When the user changes a parameter in any of the module user interfaces, the module is re-executed, and all changes are automatically propagated to all connecting modules. The user is freed from worrying about details of data dependencies and data file formats. Changes can be made without stopping the computation, thus "steering" the computational process. When other changes are made, the computations will be cancelled and automatically re-started, making the computer efficient as a "computational workbench."

## A. SCIRun - Dataflow System and Visual Programming

Designing an environment to allow the steering of complex scientific models is an enormous multi-faceted problem, one which requires attention in many different areas, including programming of the system, exploiting parallelism, and interacting with the human user.

**Programming SCIRun:** A network of modules in SCIRun forms a dataflow program. The system is programmed visually, with pre-packaged modules connected through use of the mouse. If the system does not provide the necessary components for a particular

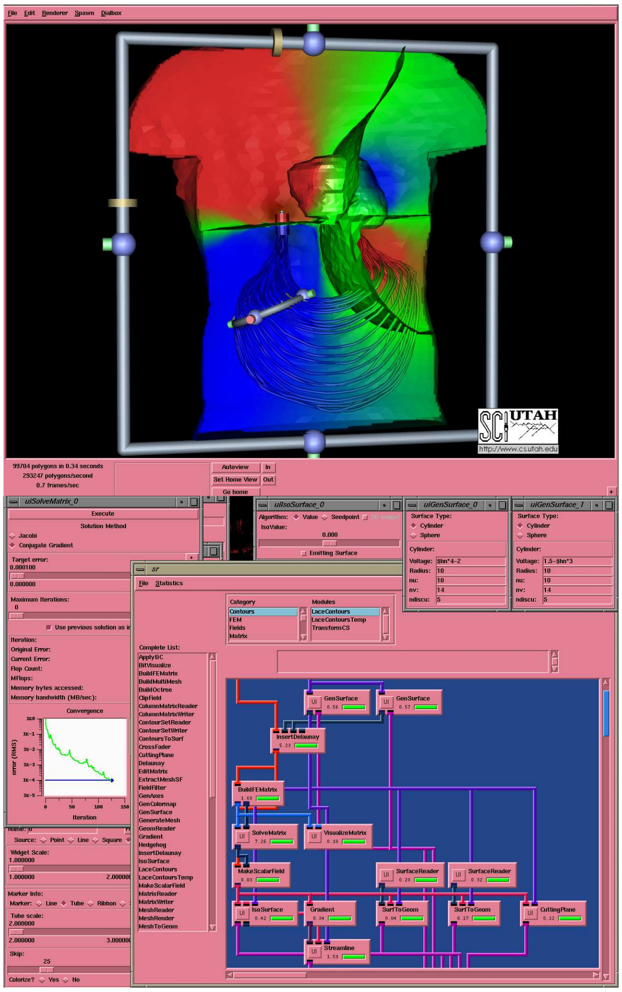

Fig. 1. An example SCIRun network, showing the dataflow programming interface, user interfaces for controlling simulation parameters, and results from an large finite element model.

task, new modules may be created by the user. When building a new module, the programmer may leverage off of existing data structures (hash tables, binary trees, linked lists, *etc.*) and utility routines (point and vector geometry, numerical integration, *etc.*). Currently, adding a new module is accomplished by implementing a C++ class.

**Parallelism:** The system is able to exploit parallelism within or between modules. Inter-module parallelism allows each module to be executed in parallel as soon as data is available on any of its input ports. Intra-module parallelism can be exploited by the module writer to achieve maximum performance for a specific algorithm. For example, a Streamline module may compute each streamline using a different processor, or may utilize an existing domain decomposition to pass the advecting particles from one processor to another. The system provides the module writer with hooks for exploiting parallel resources, but the parallel algorithms must be implemented manually by the module writer. Deciding how to allocate resources among the available parallel tasks is an open research problem that we (and many others) are still investigating. Eventually, we would like such allocations to be under the control of the user, to permit steering all aspects of the computation.

**User Interaction:** SCIRun facilitates control over many parameters, including model parameters in 3D space. While scientists are excited by this opportunity, 3D interaction presents a very complex human computer interaction problem. While we have not entirely solved these problems, we have addressed them by employing *3D widgets* [10] to assist interaction. Clear presentation of the large quantities of information produced will require further research in 2D and 3D user interface design.

# IV. Taxonomy of Steering Systems

Even though the area of computational steering is fairly young, many systems and tools exist to assist programmers and scientific researchers in tuning and running scientific codes. It would be helpful to think of these computational tools and systems within a conceptual framework in order to compare and contrast them. In the following section, we will review the work of others who previously sought to classify computational steering systems. Afterwards, we will present a cohesive taxonomy for describing computational steering systems and toolsets.

## A. Previous Classifications

Burnett, *et al.*[11] propose a taxonomy for computational steering using visual languages. Visualization systems studied vary on a continuum from post-processing through tracking to interactive visualization to steering. Interfaces presented range from a textual interface to a graphical user interface to a visual programming language interface. The authors argue for a merging of the interactive experimentation allowed by steering capabilities and the ease of use of a visual programming language for a researcher not trained in programming.

Vetter and Schwan [12] delineate two types of steering in existing systems: human-interactive steering and algorithmic steering. In human-interactive steering, a person monitors the computation and manipulates parameters of the computation while it is executing. In algorithmic steering, the computer makes decisions by monitoring information and other sources such as history files. Vetter and Schwan describe a simple feedback model for computational steering wherein output is monitored by a steering agent, either human or algorithm. The steering agent performs steering actions (which could be changes to the parameters of the computation) based on monitored inputs. They provide examples demonstrating the steering of an application's performance (load-balancing), which automatically adapts the distributed load based upon run-time statistics.

As noted by Burnett *et al.*for human-interactive steering, the mechanism of interaction affects the ease of use of the system to a scientist. Systems range from providing a textual interface from which to steer to providing a graphical interface. Of course, a visual programming language could foster the creation of a steering environment that allows the user to view the program, the simulation, and the steering mechanism potentially all at the same time. On the other hand, algorithmic steering would be programmed entirely behind the scenes, but would require more programming expertise.

While both of these classifications provide insight into differing tools and applications for simulation steering, they provide orthogonal views. Burnett's work focuses on the level of steering and the visual interface while Vetter's classification is based upon whether the steering process could be automated. Next, we will review existing tools for simulation steering and present a different taxonomy that attempts to highlight the richness of a simulation steering environment or toolset.

## B. Some Existing Tools for Steering

**Lightweight Steering: Scripting Languages and Wrappers:** Beazley and Lomdahl [13] demonstrate the use of a lightweight method of steering a very large-scale molecular dynamics simulation. Using a Simplified Wrapper Interface Generator (SWIG) to wrap existing simulation codes, a scientific researcher can easily build a scripting language interface, such as Tcl/Tk or Python, for steering a computation. Their work highlights the ease of converting existing scientific codes into a form in which they can be glued together by a control language. Then, the researcher monitors and manipulates the computation or simulation using scripting commands. Clearly, this method requires knowledge of how to program in scripting languages and does not explicitly constitute a steering toolkit.

**CUMULVS:** The CUMULVS library [14], developed at ORNL, acts as a middle layer between PVM applications and existing visualization packages such as AVS. After initializing a viewer, the application programmer can provide a list of parameters to be adjusted on-the-fly in a CUMULVS steering initialization procedure call. Separate procedure calls are used for altering scalar or vector parameters from within the application. CUMULVS supports multiple viewers viewing the same running application to assist collaborations. An interesting checkpointing capability for rolling back and restarting a failed program run has the potential to allow cross-platform migration and heterogeneous restart of an application.

**Progress and Magellan:** The Progress Toolkit [15], developed at the Georgia Institute of Technology, assists application programmers in developing steerable applications. Programmers instrument their applications with library calls, using "steerable objects," which can be altered at runtime through the use of the Progress runtime system. Steerable objects include sensors, actuators, probes, function hooks, complex actions, and synchronization points. Progress uses a client/server program model.

Developed by the same group, the Magellan Steering System [16] is derived from the Progress system, and extends the steering clients and steering servers model used in the initial system. This system uses a specialized specification language, ACSL, which provides commands for monitoring and steering using probes, sensors and actuators. However, application codes still must be instrumented with these commands in order to utilize the steering capabilities of this system. These systems have been used for Molecular Dynamics simulations.

Both systems are layered on top of the Falcon system [17], also developed at GIT, which monitors a running program, capturing information ranging from a single program variable, much as a debugger would, to complex expressions. It also permits the monitoring of performance data, with interfaces to visualization systems, such as Iris Explorer. However, decisions about which steering actions to take are based on previously encoded routines stored in a steering event database located on a steering server.

**VASE:** The Visualization and Application Steering Environment [18] (VASE), from the Center for Supercomputing Research and Development at UIUC, provides a toolset for interactive visualization and steering in a distributed environment. The VASE user model identifies three distinct roles: an application developer who writes the scientific codes; a configurer who sets up the distributed environment (including interprocess communication); and an end user (or researcher) who uses and steers the application. Steering is accomplished through the use of *steerable locations* (programmer-defined breakpoints), altering the values of variables and parameters, and adding programming statements and scripts as the computation proceeds. VASE uses a control-flow programming model, which is displayed to the end user to guide steering. VASE allows algorithm refinement through the use of script modification at run-time. Thus, the steering process can modify not only the computational parameters and performance characteristics but also the actual code.

**Pablo:** The Pablo performance environment [19], also designed at UIUC, provides library routines for instrumenting source code to extract performance data as the code executes. This system follows the Falcon model of utilizing sensors to collect information from the executing code, and altering system characteristics or parameters through the use of actuators. It seeks to tune the performance of running applications as they execute. Two different models for performance-directed adaptive control (or performance steering) are discussed: closed-loop adaptive control and interactive adaptive control. First, a neural network classifies file access patterns qualitatively in order to change the file policy on-the-fly, varying cache size and cache block replacement policy as needed by the executing code. Secondly, to enable human-interactive steering, Pablo developers argue for the use of an immersive environment, specifically the Avatar virtual environment prototype [20] built at the UIUC/NCSA.

## C. A Taxonomy for Steering

After examining the tools presented above (among others), we identified three distinct types of steering in current systems: application steering, algorithm refinement, and performance steering. Application steering refers to the capability to modify the computational process through parameter changes, mesh modifications, or other changes that affect the computational aspects of the simulation. Steering by algorithm refinement allows the underlying code to be modified or refined at runtime. Performance steering focuses on changing computational resources that affect the simulation performance such as load balancing, I/O, cache strategies or other performance related modifications.

Similarly, there is a continuum of interaction strategies from textual to visual programming that provide means for a user to interactively steer the computation. It should be noted that any steering modification could also be accomplished through automated means (*i.e.,*requiring no end user interaction), as described in some of the systems above. Figure 2 places these systems within this steering taxonomy. Arrows extend from each system to show a range of interaction possibilities.

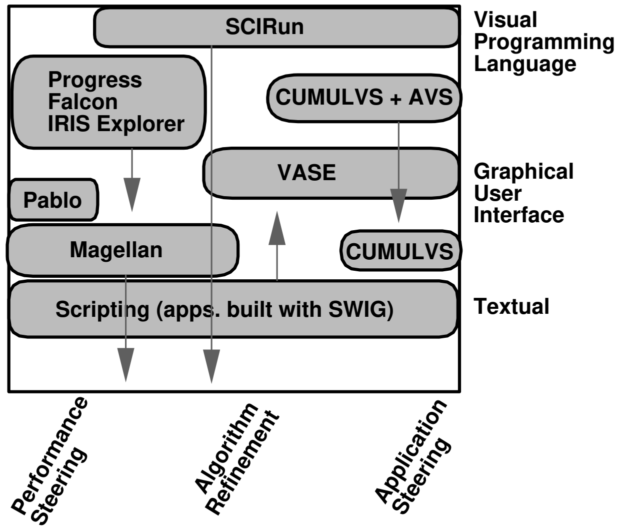


Fig. 2. Taxonomy of Steering Systems and Tools.

SCIRun was designed to allow many forms of interaction for scientists within a stand alone system. It provides application steering and algorithm refinement, but currently provides little true performance steering. Scripting, while mostly limited to text-only manipulation, spans the gamut of steering functionality, permitting performance steering, algorithm refinement, and application steering. Other systems fill a steering niche, such as Pablo's focus on performance steering. Finally, some systems, such as Progress, Magellan, Falcon and CUMULVS, provide a range of steering functionality and many forms of interaction when coupled with a visualization system such as AVS or IRIS Explorer.

## V. Advantages of Problem Solving Environments for Steering

The usefulness of computational steering tools and systems need not be argued. However, the mechanisms for implementing these tools differ tremendously as we illustrate in the steering taxonomy. Many times, steering mechanisms limit the steering to either modifications during the algorithm development phase or during the modeling and computational cycle but inhibit steering for all phases. Problem Solving Environments (PSEs) extend capabilities by allowing similar steering mechanisms to be exploited during all phases of development, application, and performance steering. They also allow the same visualization and analysis tools to be used during all phases.

A Problem Solving Environment attempts to integrate a domain-specific library with a high-level user interface, consisting of a very high-level language and a graphical interface, through the use of software infrastructure. Problem solving in scientific computation typically involves symbolic computation, numeric computation, and visualization. Thus, many PSEs, such as MatLab, Mathematica, Maple, and ELLPACK [21] integrate numerical libraries with visualization post-processing. In most PSEs, the flow of data is unidirectional, inhibiting steering of the computation. An extensive list of PSEs can be found on-line [22].

An integrated problem solving environment provides a complete set of tools for a scientist to solve a class of problems. In this context, computational steering can be a versatile tool for making changes in models, for developing new algorithms, for visualizing and analyzing results, and for tuning the performance of an application. Programming tools may be a necessary evil of the process, but the intent is for the PSE to help the scientist accurately solve a problem in a minimum amount of time. Nonetheless, scientists typically expend significant energy on programming, and they want answers to "what-if" questions for things like cache performance, multiprocessor communication patterns, memory usage, and so forth.

Steering a large scientific application involves much more than slapping a graphical user interface on a few parameters. Several of the papers mentioned above have suggested excellent methods for extracting information from running programs, for injecting updates back into the program, and for managing these changes. We argue that these techniques will be most effective when used in a highly integrated environment, where data can be shared among the various computing and visualization tasks. For our research, this integrated environment is called SCIRun.

SCIRun employs a blend of object-oriented (C++), imperative (C and Fortran), scripted (Tcl) and visual (the SCIRun Dataflow interface) languages to build this interactive environment. The basic SCIRun system provides an optimized dataflow programming environment, a sophisticated C++ data model library, resource management and development features. SCIRun modules implement components for computational, modeling and visualization tasks.

### A. Steering in a Dataflow System

Computational steering has been implemented in several dataflow environments [11], [12], [14]. The naive approach utilized by these systems allows modifying computation based upon outputs from the modules (inter-module steering). As Vetter and Schwan point out, there are three basic problems with this approach: module granularity is crucial; modifications require re-computation; and modifications are limited to the number of module inputs [12]. Most dataflow-based computational steering tools suffer from these limitations. The first and third limitation are related. If one provides too few modules within the dataflow network (very coarse granularity), steering options are limited since changes are based on connections between the modules. It is obvious that the number of module inputs limit the options for steerable parameters. The second limitation relates to whether intermediate results can be retained or not. If upstream modules modify their output, downstream modules without internal state must recompute since their inputs will change.

SCIRun removes these limitations by expanding methods for implementing steering. Four different methods are used to implement steering in the dataflow-oriented SCIRun system:

**Feedback loops in the dataflow program**. For a time-varying problem, the program usually goes through a time-stepping loop with several major operations inside. The boundary conditions are integrated in one or more of these operations. If this loop is implemented in the dataflow system, then the user can make changes in those operators that will be integrated on the next trip through the loop. This is the typical inter-module steering.

**Cancellation**. When parameters are changed, the module can choose to cancel the current operation. For example, if boundary conditions are changed, it may make sense to cancel the computation and focus on the new solution. Cancelling is sensible when solving elliptic boundary value problems, since the solution does not depend on any previous solution.

**Direct lightweight parameter changes**. An iterative matrix solver module allows the user to change the target error even while the module is executing. The parameter change does not pass a new token through the dataflow network, but simply changes the internal state of the module, effectively changing the definition of the operator rather than triggering a new dataflow event. This allows intra-module steering rather than just inter-module steering. This technique allows changes to take place outside of the dataflow stream.

**Retained state across module firings**. Modules are not required to be stateless. They may use knowledge from previous iterations to optimize the currently executing operations. For example, the matrix solver module uses the solution vector from the previous execution as the initial guess for an iterative solution method. When the changes made to the system are small, the solver will converge very quickly, similar to a time-dependent system exploiting temporal coherence by using the previous time-step as the initial guess for the next time step. A more complex example of retained state is a Delaunay triangulation module that only re-meshes local regions around boundaries that have moved since the previous iteration.

These methods provide the mechanisms whereby computational parameters can be changed during program execution. This technique creates a much richer set of steerable parameters than previous systems.

## B. Steering Optimizations

To accommodate the large datasets required by high resolution computational models, we have optimized and streamlined the dataflow implementation. These optimizations are made necessary by the limitations many scientists have experienced with currently available dataflow visualization systems [23].

**Progressive Refinement:** Unfortunately, because of memory and speed limitations of current computing technologies, it will not always be possible to complete these large scale computations at an interactive rate. To maintain interactivity, the system displays intermediate results as soon as they are available. Such results include partially converged iterative matrix solutions, partially adapted finite element grids, and incomplete streamlines or isosurfaces. In this way, an engineer or scientist can watch a solution converge and decide, based on the results observed, to make changes and start over or allow the simulation to continue to full convergence.

**Exploiting Interaction Coherence:** Another common interactive change consists of moving and orienting portions of the geometry. Because of the nature of this interaction, surface movement is apt to be restricted to a small region of the domain. As mentioned above, state can be maintained across module executions to allow incremental updates to the results.

**Data structure management:** A naive implementation of the dataflow paradigm might use the interconnection structure to make copies of the data. SCIRun uses shared copies of application data to allow the computation and visualization algorithms to work without making copies of the data. These shared regions may allow synchronized or unsynchronized access to common data. Resources are managed with a simple reference counting scheme.

Through coupling these techniques, we are able to introduce some degree of interactivity into a process that formerly took hours, days or even weeks. Most of the optimizations come from the reduced requirement for human intervention, translation programs and large file I/O. While some of these techniques (such as displaying intermediate results) will add to the computation time of the process, we attempt to compensate by providing optimizations (such as exploiting interaction coherence) that are not available with the old "data file" paradigm.

### C. Responsibilities of a Problem Solving Environment

A problem solving environment should be an efficient tool for solving all aspects of a problem. It should provide flexible modeling, visualization and computational components, but it will never provide a comprehensive set. As a result, it should also provide facilities for implementing, debugging and tuning new components. The scientist should be able to use these same tools throughout the process - during development, debugging, tuning, production and publication. To provide an efficient environment for developing and controlling scientific computations, the problem solving environment assumes responsibilities.

The first responsibility is to provide a flexible interface for reusing modeling, computational and visualization components. SCIRun accomplishes this through a visual programming interface that allows a scientist to compose appropriate tools for analyzing and visualizing various data (including end results, intermediate results, and even debug data).

A problem solving environment should also be responsible for providing an appropriate development environment for PSE components. Development includes debugging and performance tuning. Traditional debuggers are typically not efficient at dealing with the amount of data that scientific programs produce, so SCIRun allows the scientist to use the same visual analysis tools to examine and probe intermediate results. SCIRun also provides visualizations of memory usage, module CPU usage reports, and execution states. The development environment is further enhanced through cooperation with a traditional debugger, which allows the user to closely examine internal data structures when a module fails. SCIRun employs dynamic shared libraries to allow the user to recompile only a specific module without the expense of a complete re-link. Another SCIRun window contains an interactive prompt that gives the user access to a Tcl shell that can be used to interactively query and change parameters in the simulation.

Another responsibility of a problem solving environment is to assure the efficient use of system resources. In a sophisticated simulation, each of the individual components (modeling, mesh generation, nonlinear/linear solvers, visualization, *etc.*) typically consumes a large amount of memory and CPU resources. When all of these pieces are connected into a single program, the potential computational load is enormous. To use the resources effectively, SCIRun adopts a role similar to an operating system in managing these resources. SCIRun manages scheduling and prioritization of threads, mapping of threads to processors, inter-thread communication, thread stack growth, memory allocation policies, and memory exception signals.

Steering tools and environments, such as Magellan and Pablo, that focus on performance steering and algorithm refinement, address some of these issues. They provide mechanisms for performance tuning that can either be controlled by the user/developer or automated based upon performance statistics. However, they do not provide a rich set of components for computational steering of an application. By having an integrated steering environment for both developing and running an application, the user/developer has the capability to easily migrate from development to production. Furthermore, steering modifications that affect the performance can be more easily understood if discovered in an interactive setting.

### D. Requirements of the Application

A problem solving environment provides a framework for constructing and executing steerable scientific and engineering applications. However, the application programmer must assume the responsibility of breaking up an application into suitable components. In practice, this modularization is already present inside most codes, since modular programming has been preached by software engineers as a sensible programming style for years.

More importantly, it is the responsibility of the application programmer to ensure that parameter changes make sense with regard to the underlying physics of the problem. In a CFD simulation, for example, it is not physically possible for a boundary to move within a single timestep without a dramatic impact on the flow. The application programmer may be better off allowing the user to apply forces to a boundary that would move the boundary in a physically coherent manner. Alternatively, the user could be warned that moving a boundary in a non-physical manner would cause gross errors in the transient solution.

## VI. Conclusions

SCIRun attempts to overcome the artificial distinctions between scientific computing, scientific visualization, and computational steering. Many visualization tasks are scientific computing problems themselves, hence are further candidates for steering. The primary goal of SCIRun is to provide the scientist with a comprehensive environment with interfaces to control and interact with the simulation at both application and system levels, and to use scientific visualization in all aspects of the problem. This control can be implemented with the best available techniques, such as those reviewed above. By integrating computational and visualization components, SCIRun avoids the transfer of large datasets to a separate visualization process. In addition, the scientist can use the same visualization tools in the development stages, the performance tuning stages, the production stages, and even the publication stages of the scientific application.

At the application level, the mathematical requirements of steering are as important as the program implementation, yet those requirements receive much less focus. For example, it is very easy for a "steerer" to input new data that represents a mathematically invalid or physically impossible transition. Opponents of steering point at these occurrences and question the validity of the results from such a system. Methods of integrating changes in a scientifically meaningful manner need considerable investigation. While it is important to have complete control of all parameters while debugging a simulation, production simulations should have either tight requirements for valid steering input or a flexible system to help the scientist assess the validity of the transformation.

Computational steering systems are best implemented as a part of an integrated problem solving and development environment. With limited programming effort, computational scientists should be able to interactively create, control, execute, and visualize complex scientific simulations.

## VII. Acknowledgments

This work was supported in part by the NSF, NIH, DOE, and the Utah State Centers of Excellence.

## References

[1] C.R. Johnson, R.S. MacLeod, and M.A. Matheson, "Computational medicine: Bioelectric field problems", *IEEE COMPUTER*, pp. 59–67, Oct., 1993.

[2] T. De Fanti et al, "Special issue on visualization in scientific computing", *Computer Graphics*, vol. 21, no. 6, Nov. 1987.

[3] W. Gu, J. Vetter, and K. Schwan, "An annotated bibliography of interactive program steering", *Georgia Institute of Technology Technical Report*, 1994.

[4] S.G. Parker, D.M. Weinstein, and C.R. Johnson, "The SCIRun computational steering software system", in *Modern Software Tools in Scientific Computing*, E. Arge, A.M. Bruaset, and H.P. Langtangen, Eds., pp. 1–44. Birkhauser Press, 1997.

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

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

[7] C. Upson and et al., "The application visualization system: A computational environment for scientific visualization", *IEEE Computer Graphics & Applications*, vol. 9, no. 4, pp. 30–42, July 1989.

[8] B. Lucas and et al., "An architecture for a scientific visualization system", in *Proceedings of Visualization '92*. 1992, pp. 107–114, IEEE Press.

[9] C. Willams, J. Rasure, and C. Hansen, "The state of the art of visual languages for visualization", in *Proceedings of Visualization '92*. 1992, pp. 202–209, IEEE Press.

[10] R. Zeleznik and et al., "An interactive 3d toolkit for constructing 3d widgets", *Computer Graphics (Proceedings of SIGGRAPH '93)*, pp. 81–84, July 1993.

[11] M. Burnett, R. Hossli, T. Pulliam, B. VanVoorst, and X. Yang, "Toward visual programming languages for steering scientific computations", *IEEE Computational Science and Engineering*, vol. 1, no. 4, pp. 44–62, 1994.

[12] J. Vetter and K. Schwan, "Models for computational steering", in *Proceedings of the Third International Conference on Configurable Distributed Systems*, 1996.

[13] D.M. Beazley and P.S. Lomdahl, "Controlling the data glut in large-scale molecular-dynamics simulations", *Computers in Physics*, vol. 11, no. 3, pp. 230–238, May/June 1997.

[14] G.A. Geist, J.A. Kohl, and P.M. Papadopoulos, "Cumulvs: Providing fault-tolerance, visualization and steering of parallel applications", *SIAM*, Aug. 1996.

[15] J. Vetter and K. Schwan, "Progress: A toolkit for interactive program steering", in *Proceedings of the 24th Annual Conference of International Conference on Parallel Processing*, 1995, pp. 139 – 142.

[16] J. Vetter and K. Schwan, "High performance computational steering of physical simulations", in *Proceedings of the 11th International Parallel Processing Symposium*. Apr. 1997, Geneva, Switzerland.

[17] W. Gu, G. Eisenhauer, E. Kramer, K. Schwan, J. Stasko, and J. Vetter, "Falcon: On-line monitoring and steering of large-scale parallel programs", in *Proceedings of the 5th Symposium of the Frontiers of Massively Parallel Computing*. Feb. 1995, pp. 422 – 429, ACM.

[18] D.J. Jablonowski, J.D. Bruner, B. Bliss, and R.B. Haber, "Vase: The visualization and application steering environment", in *Proceedings of Supercomputing '93*. 1993, pp. 560 – 569, IEEE Computer Society Press.

[19] D.A. Reed, C.L. Elford, T.M. Madhyastha, E. Smirni, and S.E. Lamm, "The next frontier: Interactive and closed loop performance steering", in *Proceedings of the 25th Annual Conference of International Conference on Parallel Processing*, 1996.

[20] D.A. Reed, K.A. Shields, L.F. Tavera, W.H. Scullin, and C.L. Elford, "Virtual reality and parallel systems performance analysis", *IEEE Computer*, pp. 57 – 67, Nov. 1995.

[21] J.R. Rice and Boisvert R.F., *Solving Elliptic Problems using ELLPACK*, Springer-Verlag, 1984.

[22] ", Problem Solving Environments - Projects, Products, Applications and Tools: http://www.cs.purdue.edu/research/cse/pses/research.html.

[23] B. Ribarsky and et al., "Object-oriented, dataflow visualization systems—A paradigm shift?", in *Proceedings of Visualization '92*. 1992, pp. 384–388, IEEE Press.