Applications in Computational Medicine using SCIRun: A Computational Steering Programming Environment

Christopher R. Johnson and Steven G. Parker
Department of Computer Science
University of Utah
Salt Lake City, Utah 84112
Email: crj@cs.utah.edu and sparker@cs.utah.edu
URL: http://www.cs.utah.edu/~ sci

Abstract

In this paper we describe applications of the SCIRun system to large scale problems in computational medicine. SCIRun is a scientific programming environment that allows the interactive construction and steering of large scale scientific computations. Using this "computational workbench," scientists and engineers 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. As opposed to the typical "off-line" simulation mode - in which the scientist manually sets input parameters, computes results, visualizes the results via a separate visualization package, then starts again at the beginning - SCIRun "closes the loop" and allows interactive steering of the design and computation phases of the simulation.

Introduction

The current computational engineering and science modeling process is a familiar one – create, or make modifications to, a geometric model, input initial conditions and/or boundary conditions, numerically approximate solutions to the governing equation(s), store data off to disk and/or tape, visualize the results via a separate visualization package, decide upon appropriate changes, and repeat the whole process *ad nauseam*. Usually, the modifications made to the model, input parameters, numerical and computing considerations, and visualization parameters are accomplished using separate in-line, command based instructions, and each change made in the model and/or input parameters causes all of the other steps to be repeated. Even for experienced scientists who may employ scripts and conversion programs to aid them in the task, the process is anything but streamlined. Ideally, scientists and engineers would be provided with a system in which all these computational components were linked, so that all aspects of the modeling and simulation process could be controlled graphically within the context of a single application program. While this would be the preferred *modus operandi* for most computational scientists, it is not the current standard of scientific computing because *the creation of such a program is an extraordinarily difficult task.*

To reduce the time the scientist spends in the current modeling process and to provide a tool for exploration of computational science and engineering problems, we have developed SCIRun* [1, 2]. SCIRun is a framework 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. 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.

The SCIRun system implements a new model for scientific computing. This model relies on modern computing technologies, such as graphical user interfaces and 3D graphics, to provide a less cumbersome but equally expressive format to investigate complex scientific problems. The increased flexibility attempts to provide a computational workbench for scientific computing. Within this workbench new experiments are formed, new methods are explored, and tedious coding is kept to a minimum.

Computational steering has been defined as "the capacity to control the execution of long-running, resource-intensive programs" [3]. In the field of computational science, we apply this concept to link visualization with geometric design and computational phases to interactively explore (steer) a simulation in time and/or space. As knowledge is gained, it can be used to change the input conditions and/or other parameters of the simulation. Implementation of a computational steering framework requires a successful integration of the many aspects of scientific computing, including geometric modeling, numerical analysis, and scientific visualization, all of which need to be effectively coordinated within an efficient computing environment.

The construction of SCIRun has been motivated by our work in applying the computational paradigm to large scale problems in medicine - *computational medicine*. Over the past several years we have developed numerous techniques to solve computationally intensive problems in medicine focusing primarily on bioelectric field problems,

*Pronounced "ski-run." SCIRun derives its name from the Scientific Computing and Imaging (SCI) research group at the University of Utah.

such as (1) interactive programs to construct, manipulate, and display large scale, three-dimensional surface and volumetric meshes [4, 5, 6, 7]; (2) three-dimensional, adaptive, finite element programs for solving elliptic partial differential equations with general boundary conditions and source terms [4, 8, 9, 10]; (3) programs that solve the resulting large system of equations and simultaneously apply global and regularization to ill-posed inverse problems [4, 11]; and (4) scientific visualization packages for displaying and interpreting the results [5, 6, 7, 12]. Because our work involves a large problem size, much of the software we have developed has been designed to take advantage of parallel processing via high performance architectures.

Large scale bioelectric field problems occur in a wide variety of biomedical research areas ranging from models of single cells through models of whole organs to models which incorporate partial human structures. Solutions of these problems aid in analysis and design of effective defibrillation and cardiac pacing systems, in the noninvasive detection and location of cardiac arrhythmias, and in imaging technologies such as electrical impedance tomography (EIT) and applied potential tomography (APT). The last application (APT) is clinically useful for localizing and analyzing spontaneous brain activity in epileptics and holds promise as a noninvasive imaging technique for predicting detailed bioelectric information on the heart or brain from signals measured on the body or scalp - the so-called generalized inverse problem in electrocardiography (ECG) and and electroencephalography (EEG) problems, respectively.

Applications in Cardiology and Neuroscience

Here we address the application of SCIRun to two bioelectric field problems in medicine, simulation of cardiac defibrillation and simulation of temperal lobe epilepsy.

Every year, approximately 500,000 people die suddenly because of abnormalities in their hearts' electrical system (cardiac arrhythmias) and/or from coronary artery disease. While external defibrillation units have been in use for some time, their use is limited because it takes such a short time for a heart attack victim to die from insufficient oxygen to the brain. Lately, research has been initiated to find a practical way of implanting electrodes within the body to defibrillate a person automatically upon onset of cardiac fibrillation. Because of the complex geometry and inhomogeneous nature of the human thorax and the lack of sophisticated thorax models, most past design work on defibrillation devices has relied on animal studies. We have constructed a large scale model of the human thorax, the Utah Torso Model [4, 5, 6, 7], for simulating both the endogenous fields of the heart and applied current sources (defibrillation devices). Using these computer models, We are also able to simulate the multitude of electrode configurations, electrode sizes, and magnitudes of defibrillation shocks. Given the large number of possible external and internal electrode sites, magnitudes, and configurations, it is a daunting problem to computationally test and verify various configurations. For each new configuration tested, geometries, mesh discretization levels, and a number of other parameters must be changed.

Excitation currents in the brain produce an electrical field that can be detected as small voltages on the scalp. By measuring changes in the patterns of the scalp's electrical activity, physicians can detect some forms of neurological disorders. Electroencephalograms, EEGs, measure these voltages; however, they provide physicians with only a snapshot of brain activity. These glimpses help doctors spot disorders but are sometimes insufficient for diagnosing them. For the latter, doctors turn to other techniques; in rare cases, to investigative surgery.

Such is the case with some forms of epilepsy. To determine whether a patient who is not responding to medication has an operable form of the disorder, known as temporal lobe epilepsy, neurosurgeons use an inverse procedure to identify whether the abnormal electrical activity is highly localized (thus operable) or diffused over the entire brain.

To solve these two bioelectric field problems in medicine, we are using SCIRun to design internal defibrillator devices and measure their effectiveness in an interactive graphical environment. Similarly we are using SCIRun to develop and test computational models of epilepsy.

Using SCIRun, scientists and engineers are able to design internal defibrillation devices and source models for the epileptic foci, place them directly into the computer model, and automatically change parameters (size, shape and number of electrodes) and source terms (position and magnitude of voltage and current sources) as well as the mesh discretization level needed for an accurate finite element solution. Furthermore, engineers can use the interactive visualization capabilities to visually gauge the effectiveness of their designs and simulations in terms of distribution of electrical current flow and density maps of current distribution.

Mathematically, these problems are governed by variations of the generalized Poisson's equation (or its homogeneous counterpart, Laplace's equation) for electrical conduction in the physical domain Ω [13]:

$$\nabla \cdot \sigma \nabla \Phi = -I_V \tag{1}$$

where I_V is the electrical current source and σ is an electrical conductivity tensor. The electrostatic potential Φ is subject to the boundary conditions:

$$\Phi = \Phi_0 \;\text{ on }\; \Gamma_1 \;\text{ and }\; \sigma \nabla \Phi \cdot \mathbf{n} = 0 \;\text{ on }\; \Gamma_2 \tag{2}$$

where Γ_1 is the surface of the internal defibrillator electrodes, and Γ_2 is the surface of the torso (scalp). Φ_0 specifies a Dirichlet boundary condition on this surface. With $\mathbf{n}$ as the unit surface normal vector, $\sigma \nabla \Phi \cdot \mathbf{n}$ represents the current flow normal to the surface of the torso (scalp), which is zero for the insulated boundary.

Once the electrostatic potentials are known, one can calculate the current density $\mathbf{J}$ according to:

$$\mathbf{J} = -\sigma \nabla \Phi. \tag{3}$$

For the defibrillation problem, electrodes are either implanted internally or applied directly to the chest in order to deliver sufficient electric energy to stop the irregular heart rhythms that signify a fibrillating heart [14, 15]. Mathematically, this can be posed as solving equations (1-3) with the voltage boundary condition applied on a portion of the torso boundary, $\Sigma \subseteq \Gamma_2$ for external defibrillation or from the surface of the defibrillation electrode(s) within the volume of the thorax for internal defibrillation.

Past (and much current) practice regarding the placement of the electrodes for either type of defibrillator has been determined by clinical trial and error. One of our goals is to allow engineers to use SCIRun to assist in determining the optimum electrode placement, size, shape, and strength of shock to terminate fibrillation by solving equations (1-3) within a detailed model of the human thorax.

For the neuroscience problem, the epileptic foci is represented as a set of idealized dipole sources situated in the temporal lobe. Using a model of the human skull and brain, the direct EEG problem is posed by solving equations (1-3) for the voltage and current distribution within the brain and upon the surface of the scalp. For the inverse EEG problem, measured scalp voltages are used as the inputs and equations (1) is solved for the source currents I_V. We are currently using SCIRun to investigate the direct and inverse EEG problems as well as using SCIRun as an interactive modeling and visualization tool.

Components of the Problem

Geometric Modeling

In most computational engineering and science applications, much geometric modeling must take place prior to simulation and visualization, as is the case in our medical application. 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.

To solve the bioelectric field problems associated with equations (1-3), we have constructed torso and head models from MRI (magnetic resonance imaging) scans. Two semi-automatic segmentation algorithms are used to classify the relevant tissues [16, 17]. We then utilize two- and three-dimensional Delaunay triangulation algorithms to construct three-dimensional surfaces and/or volumetric meshes [10, 18, 19]. Geometric postprocessing consists of assigning each tetrahedron an electrical conductivity tensor.

Construction of the geometric model is often one of the most time consuming aspects of the modeling and simulation process. 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 a computational steering framework, the goal is to change geometric features of the model or the spatial discretization of the solution domain in an interactive manner. Ideally, the user receives some degree of feedback on the calculation almost immediately and is allowed to change such input boundary conditions as, for example, 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 a framework allows more immediate access to simulation results and significantly reduces the time spent in making simulation and modeling design changes.

Numerical Analysis

Because of the geometric complexity of the solution domain (i.e. the human body) and the anisotropic nature of some of the inhomogeneities (i.e. muscle) in our problems, the finite element (FE) method was the preferred choice to numerically approximate the solutions to equations (1-3). Application of the FE method yields a linear system, $\mathbf{A}\mathbf{\Phi} = \mathbf{b}$, where $\mathbf{A}$ is sparse, symmetric, and positive definite and on the order of $N = 25,000 - 1,500,000$ depending on the level of mesh refinement, on the level of interactiveness required by the user, and on the specific goals of the study [4, 5].

Due to the quasi-static nature of the bioelectric field problems we are solving [13], we are primarily interested in *spatial* steering for this application. Spatial steering involves controlling the discretization level of the geometry, typically in terms of the accuracy of the finite element model. We have implemented an adaptive method to automatically refine and de-refine a finite element mesh based upon *a posteriori* error estimates of the finite element approximation[20].

Within SCIRun, the engineer can interactively decide upon the level of accuracy used for a given simulation based upon *a priori* design criteria. Upon onset of the simulation, the engineer 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). The engineer then decides if she/he would like to continue the computation using a more (or less) refined level of discretization. Alternatively, if the engineer decided from the initial results that there needed to be change in the input conditions, he/she would make the changes, thus restarting the computation.

Scientific Visualization

Certainly, effective interpretation of computer simulations depends upon the visualization of the data. Traditionally, the visualization phase has been an entirely separate process from the computational phase. Computations are usually stored off to disk and/or piped into a separate visualization software package once all computations are completed. Furthermore, many scientists have 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 the SCIRun, visualization is an integral part of the computational and geometrical modeling phases and is used to navigate the user through the data and/or to help the user interact with the data to modify the input parameters or geometric design. 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.

To date we have developed several scientific visualization systems [5, 6, 7, 12]. One is an interactive three-dimensional GL based graphics system for viewing surface and volume geometry and simulation results. For higher quality ray-traced rendered images, we have developed a batch-mode graphics system which exploits parallel processors and/or clusters of workstations. Recently, we have developed a differential volume rendering module which allows nearly real-time rendering of three-dimensional flows for models with hundreds of thousands of elements [21]. For SCIRun, we have combined many of the components of these visualization systems with our computational and geometrical design systems to produce an integrated method for interactively designing and exploring computational science and engineering models.

The SCIRun system described below allows the unification of a large collection of algorithms for realizing the components of a scientific computing environment outlined above. Connecting these algorithms in an efficient manner accomplishes much of the computational steering goal. Although creating new modules for the system is an ongoing (and never-ending) work, current efforts have concentrated on building the tools for geometrical modeling, numerical analysis and scientific visualization which will assist in our research in computational medicine.

A Computational Steering Framework

SCIRun is a framework 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 Application Visualization Systems Inc. [22], Iris Explorer from Silicon Graphics, and Visualization Data Explorer from IBM [23] have made this archetype popular for scientific visualization [24]. Our work has extended this paradigm into the realm of scientific computation.

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 with the ability to interactively steer any one phase of the process and to see the effects propagate throughout the system automatically.

As a simple example, consider a group of visualization modules within a dataflow

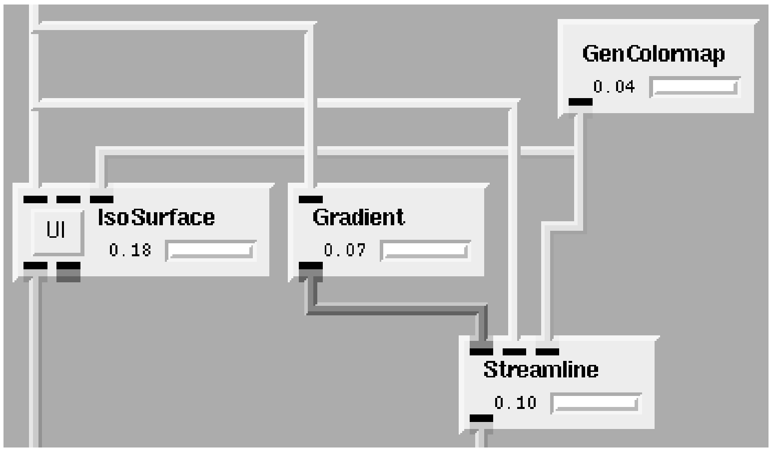

Figure 1: Dataflow Network for a Subset of Visualization Modules

network, illustrated in Figure 1. The boxes represent computational algorithms (modules) and the lines represent data pipes between the modules. Scalar field data enters through the pipe at the top, passing into a isosurfacing algorithm and a gradient computation. Streamlines are computed on the resulting gradient vector field, and colored according to the scalar values. Geometry objects for the isosurfaces and streamlines are passed out the bottom.

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. Some changes can be made without stopping the computation, thus "steering" the computational process. Other changes can be made and the computations will be cancelled and automatically re-started, making the computer effi cient as a "computational workbench."

Dataflow System and Visual Programming

Designing a framework to allow the steering of complex scientific models is an enormous multi-faceted problem, 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 "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 task, new modules may be created by the user. When building a new module, the programmer may leverage off of the data structures (hash tables, binary trees, linked lists, etc.) and utility routines (point and vector geometry, numerical integration, etc.) that already exist in the system. Currently, adding a new module is accomplished by implementing a C++ class. In the future, interfaces to C and Fortran subroutines will also be provided.

Parallelism: The system is able to exploit parallelism either inter-modularly or intra-modularly. Inter-Module parallelism allows each module to be executed in parallel, as soon as data is available on any of its input ports. Thus, many networks may allow the system to automatically execute several modules in parallel. For example, the dataflow network in Figure 1 will allow the the system to execute the Isosurface Module in parallel with the Gradient and Streamline modules. Intra-Module parallelism can be exploited by the module writer to achieve maximum performance for a specific algorithm. For example, the Streamline module may compute each streamline using a different processor. 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 which we (and many others) are still investigating. Eventually, we would like such allocations to be under the control of the user, in order that the user may steer all aspects of the computation.

User Interaction: The defibrillator design example above describes a system whereby an engineer can interactively change boundary conditions in a computational model. While engineers are excited by this opportunity, it presents a very complex human computer interaction problem, due to the difficulty of 3D interaction. While we have not entirely solved these problems, we have addressed them by employing *3D widgets* [25, 26] and alternative input devices to assist in the interaction tasks.

SCIRun Applied to Problems in Medicine

A network that can be used to model cardiac defibrillation is shown in Figure 2. A similar network is used for a forward solution in the neuroscience application. The networks consists of the following modules:

- **SurfaceReader** reads a triangulated surface definition from a file. One of these modules will read the torso boundary (body surface) geometry, and the other will read the epicardium (heart surface) geometry. In the neuroscience application, there may be several surfaces - including the scalp surface, the scalp-skull interface, the skull-cerebrospinal fluid (CSF) interface, the CSF-grey matter interface, and the grey-white matter interface.

- **GenSurface** will generate two cylindrical electrodes for the defibrillation study. Parameters in the interface allow the scientist to control the discretization of the electrodes. In the neuroscience application, these surfaces may represent dipole or other types of source configurations in the brain.

- **SurfToGeom** converts the surface definitions into displayable geometry. A flag in the user interface controls whether or not the geometry is movable. The epi-

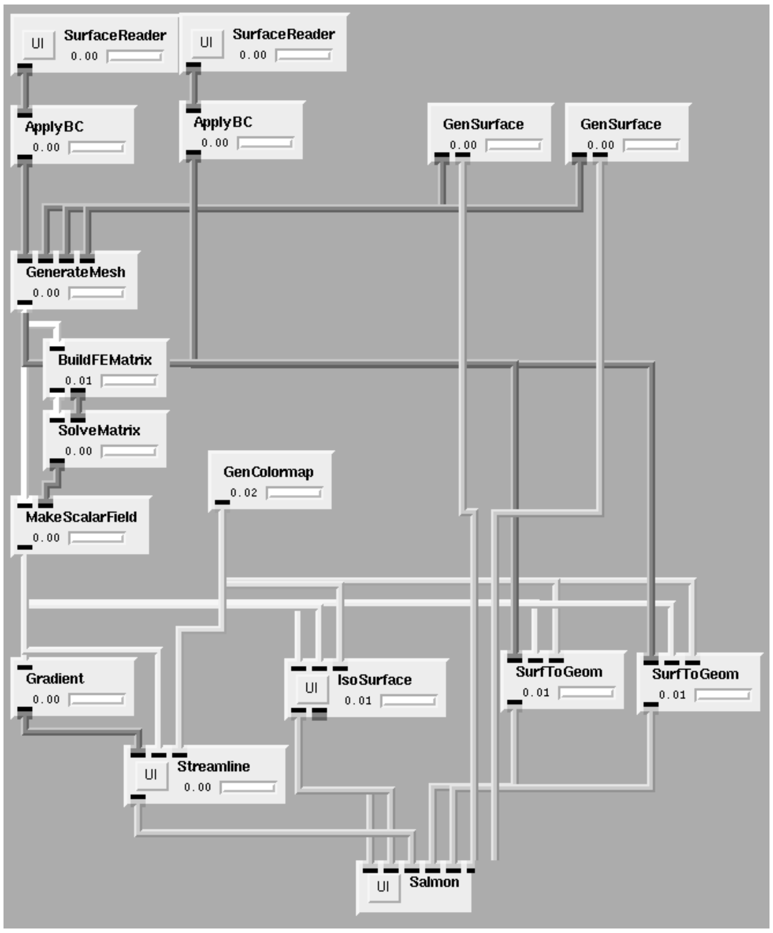

Figure 2: SCIRun Network for the Cardiac Defibrillation Problem

cardium and torso boundary should not be moved, since they correspond to physical geometry. The electrode cylinders, on the other hand, must be moved so that various placements can be tested. The SurfToGeom module provides 3D widget handles which allow the user to manipulate these surfaces directly. An optional input parameter selection will map scalar field values onto the surface. This input is attached to the epicardium and shows the voltages on the surface of the heart.

- **ApplyBC** applies boundary conditions to the various surfaces. The torso and scalp have a zero-flux Neumann boundary, in the debifrillation problem, the two electrode cylinders have Dirichlet boundary conditions corresponding to their respective voltage. In the neuroscience application, there are discrete current sources.

The voltage and current sources may be changed interactively.

- **GenerateMesh** discretizes the volume defined on the outside by the torso (scalp) and on the inside by the two voltage source electrodes. This discretization occurs after the application of the boundary conditions so that the mesh generator may optimize the mesh for the particular boundary conditions.

- **BuildFEMatrix** uses the mesh structure, the boundary conditions, and finite element theory to construct a matrix that describes the user-specified configuration. Utilizing controls on the user interface, the user may instruct the module to create a dense matrix, a band diagonal matrix or a compressed sparse-row matrix. Currently this module is able to create a matrix only for elliptic partial differential equations, but a system is being designed to allow the specification of other governing equations.

- **SolveMatrix** uses direct or iterative algorithms to find the solution to the matrix equation. We currently use a preconditioned conjugate gradient algorithm for iterative solutions. The scientist controls convergence parameters through the graphical user interface.

- **MakeScalarField** combines the solution of the finite-element matrix to the volume mesh generated by GenerateMesh. This mesh/solution combination provides a representation of the solution in terms of a scalar field of voltage values.

- **IsoSurface** allows interactive extraction of isosurfaces in the voltage field. A small sphere controls the starting point of the isosurface algorithm, and an attached 3D arrow shows the direction of the gradient. The sphere and arrow widget may be moved using the mouse to allow interactive exploration of the voltage field. Dragging on the body of the arrow moves the widget along the line defined by the gradient; dragging on the sphere allows unconstrained movement of the seed point.

- **Gradient** computes a vector field from the scalar voltage field according to equation (3). This yields another form of the solution in terms of electric current density.

- **Streamline** produces vector field lines which reveal the flow of electrical currents within the torso and brain. These field lines are analagous to massless particle traces in fluid flow fields. The streamlines are advected using a 4th order Runge-Kutta technique. The user may choose between a single streamline or a row of streamlines. Adaptation parameters and step sizes are controlled via the 2D user interface, while the positions of the particle sources are controlled with 3D widgets. The 3D widget for the single streamline consists of a small sphere which specifies the seed point for the streamline. The widget for the row of streamlines takes the form of a rake [27]. Dragging on the cylindrical shaft controls the position of the rake, while dragging on the spherical endpoints controls the orientation and the length of the rake. Finally, a small slider on the rake controls the spacing between each streamline.

- **Salmon** provides the underlying structure for viewing geometry and 3D user interaction for both viewpoint control and control of the 3D widgets described above. As the streamline module computes streamlines, or as the isosurface modules computes isosurfaces, it will send geometry (lines, triangles or other primitives) to Salmon. Salmon displays these objects geometry in a window. Some pieces of geometry are tagged as special user interface components (3D widgets). When the user drags a 3D widget with the mouse, Salmon sends messages back to the module creating the interface. This module can then update the position of the widget and any other geometry influenced by the new position. This two-way communication forms the basis for implementing 3D widgets.

Each of these modules are simple enough to be managed easily, but when they are joined together, they accomplish a very complex task. Sample visualizations from this type of network is shown in Figures 3 and 4.

Pressing the “UI” (user interface) button on any of the modules’ icons summons a pop-up window which allows the user to change various parameters for that module. For example, the Streamline module’s pop-up allows the user to specify integration parameters, display modes, 3D widget types and other information. As an example user interface, Figure 5 shows the pop-up windows for the Streamline module.

Steering Optimizations

In order 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 [28].

Data structure management: A naive implementation of the dataflow paradigm might use the pipe structure to make copies of the data. Consider the example in Figure 1. If the vector field is copied to both the Streamline and StreamSurface modules, then twice as much memory is consumed as necessary. In addition, a significant amount of CPU time is required to copy large data structures. In order to avoid these overheads, we employ *smart pointers* [29] in C++, which help reduce complexity by allowing different modules to share common data structures. When the VectorFieldReader modules create the VectorField, it sets a reference count in the vector field to zero. As the VectorFieldReader hands a copy of the vector field data structure to each of the downstream modules, the receiving module will increment the reference count. When each module is finished with the data structure, it will decrement the reference count. When the reference count reaches zero, the object will be destroyed. These reference counts are maintained automatically by C++ classes (the smart pointers) in order to reduce programmer error.

Copying the object is necessary only when a module needs to change a data structure and the reference count is greater than one (i.e. another module is also using it).

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. In order to maintain the illusion of dynamic 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 the defibrillator design example shown above, the user moves an electrode and sees some feedback almost immediately. The solution will continue to converge as long as no changes are made. If the

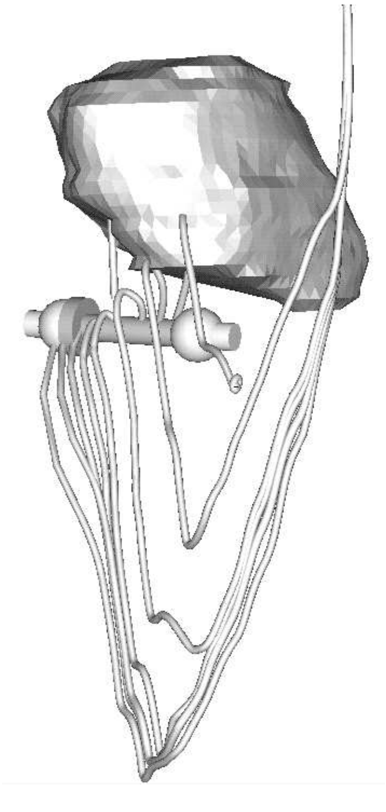

Figure 3: Visualization of a defibrillator design simulation, showing an electrode, the surface of the heart (epicardium), a 3D widget (rake), and electrical current lines (streamlines). The other electrode in the simulation is obscured by the heart. Much of the current leaving the visible electrode travels away from the heart, due to the high conductivity of blood in a nearby vessel.

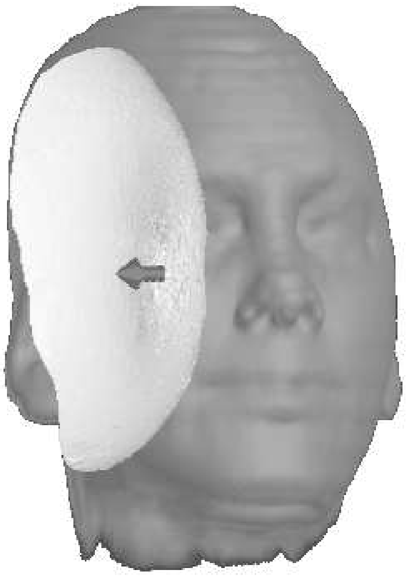

Figure 4: Visualization of a computational neuroscience simulation, showing an epileptic focus (a dipole source, indicated by the arrow) and an isopotential surface in the resulting field.

electrode is moved again, current calculations are aborted, and the new ones are started. In this way, an engineer or scientist can watch a solution converge and, based on the results observed, may either decide 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. Using the information available from both how the geometry was moved and its position prior to the move, the system is able to anticipate results and "jump start" many of the iterative methods. For example, iterative matrix solvers can be jump-started by interpolating the solution from the old geometry onto the new mesh. When changes to the model geometry are small, the resulting initial guess will be very close to the desired solution and the

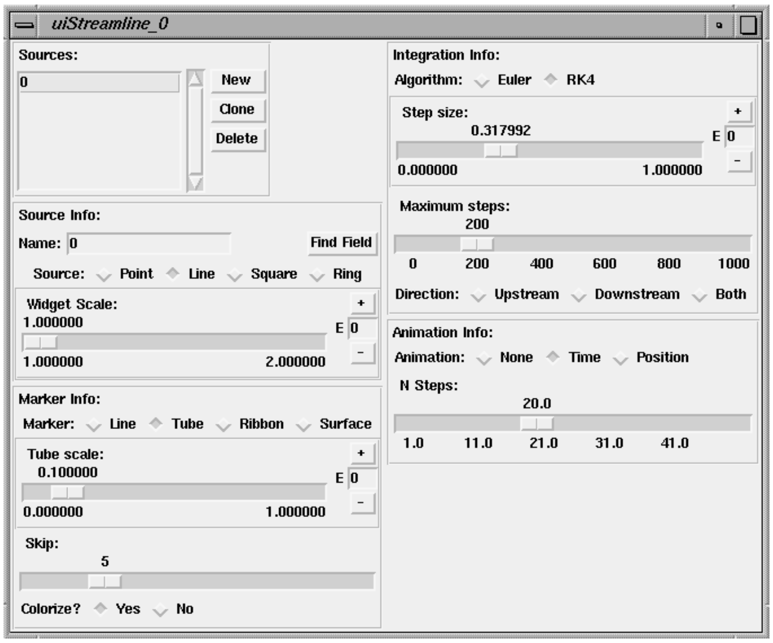

Figure 5: Pop-up Window for the Streamline Module

solver will converge rapidly. This concept is similar to exploiting temporal coherence in a time-dependent system by using the previous time-step as the guess to the next time step.

Through coupling each of these techniques, we are able to introduce some degree of interactivity into a process which formerly took hours, days or even weeks. 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) which are not available with the old "data file" paradigm.

Summary and Future Work

We have presented an overview of SCIRun applied to large scale problems in computational medicine. SCIRun is a computational steering model that allows users to "close the loop" and utilize visualization to guide (steer) the design and computation phases of a simulation. We are continuing to develop, optimize, and add flexibility to current features of our steering model. We are also adapting the computational steering model to a number of applications, including solving problems in inverse ECG and EEG, providing interactive surgery assistance for problems in neurosurgery, and addressing problems in computational fluid dynamics and environmental science.

Currently the system runs on Silicon Graphics computers, using shared memory hardware to communicate between processors in a multi-processor environment. Work is under way to port the system to new architectures. In particular, support for parallelism on workstation clusters will be integrated into the system. The dataflow network can map directly to a network of processors or machines (one or more modules per processor). Minimizing the network traffic and communication overhead for a particular dataflow network and machine configuration is an interesting problem that this extension will attempt to address.

Acknowledgements

This work was supported in part by awards from the DOE, NIH, and the NSF. The authors would like to thank K. Coles and D. Weinstein for their helpful comments and suggestions. Furthermore, we appreciate access to facilities which are part of the NSF/ARPA STC for Computer Graphics and Scientific Visualization and computational resources provided by NCSA.

References

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

[2] S.G. Parker and C.R. Johnson. Scirun: A scientific programming environment for computational steering. In *Supercomputing '95*. IEEE Press, 1995. (submitted).

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

[4] C.R. Johnson, R.S. MacLeod, and P.R. Ershler. A computer model for the study of electrical current flow in the human thorax. *Computers in Biology and Medicine*, 22(3):305–323, 1992.

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

[6] R.S. MacLeod, C.R. Johnson, and M.A. Matheson. Visualization tools for computational electrocardiography. In *Visualization in Biomedical Computing*, pages 433–444, 1992.

[7] R.S. MacLeod, C.R. Johnson, and M.A. Matheson. Visualization of cardiac bioelectricity — a case study. In *IEEE Visualization '92*, pages 411–418, 1992.

[8] C.R. Johnson, R.S. MacLeod, and M.A. Matheson. Computer simultions reveal complexity of electrical activity in the human thorax. *Comp. in Physics*, 6(3):230–237, May/June 1992.

[9] C.R. Johnson and R.S. MacLeod. Nonuniform spatial mesh adaption using a posteriori error estimates: applications to forward and inverse problems. *Applied Numerical Mathematics*, 14:311–326, 1994.

[10] J.A. Schmidt, C.R. Johnson, J.C. Eason, and R.S. MacLeod. Applications of automatic mesh generation and adaptive methods in computational medicine. In J.E. Flaherty and I. Babuska, editors, *Modeling, Mesh Generation, and Adaptive Methods for Partial Differential Equations*. Springer-Verlag, 1994 (to appear).

[11] C.R. Johnson and R.S. MacLeod. Inverse solutions for electric and potential field imaging. In R.L. Barbour and M.J. Carvlin, editors, *Physiological Imaging, Spectroscopy, and Early Dectection Diagnostic Methods*, volume 1887, pages 130–139. SPIE, 1993.

[12] R.S. MacLeod, C.R. Johnson, and M.A. Matheson. Visualizing bioelectric fields. *IEEE Computer Graphics and Applications*, pages 10–12, 1993.

[13] R. Plonsey and R. C. Barr. *Bioelectricity*. Plenum Press, New York, 1988.

[14] B. Lerman and O. Deale. Relation between transcardiac and transthoracic current during defibrillation in humans. *Circulation Research*, 67:1420–1426, 1990.

[15] M. Mirowski. The automatic implantable cardioverter-defibrillation: An overview. *Journal of the American College of Cardiology*, 6:461–466, 1985.

[16] H.W. Shen. Semi-automatic image segmentation: A bimodel thresholding approach. Technical report, UUCS-94-019, Dept. of CS, Univ. of Utah, 1994.

[17] D. Weinstein and C.R. Johnson. Hierarchical data structures for interactive volume visualization. *IEEE Visualization '95*, 1995 (submitted).

[18] D.F. Watson. Computing the n-dimensional Delaunay tesselation with applications to Voronoi polytopes. *Computer Journal*, 24(2):167–172, 1981.

[19] N. Weatherill and O. Hassan. Efficient three-dimensional grid generation using the Delaunay triangulation. In *Proc. of the 1st European CFD Conf.*, volume 1, 1992.

[20] F. Yu and C. Johnson. An automatic adaptive refinement and derefinement method. In *Proceedings of the 14th IMACS World Congress*, pages 1555–1557, 1994.

[21] H.W. Shen and C.R. Johnson. Differential volume rendering: A fast alogrithm for scalar field animation. In *Visualization '94*, pages 180–187. IEEE Press, 1994.

[22] C. Upson and et al. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics & Applications*, 9(4):30–42, July 1989.

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

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

[25] D. Conner and et al. Three-dimensional widgets. *Computer Graphics (Proceedings of the 1992 Symposium on Interactive 3D Graphics)*, pages 183–188, March 1992.

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

[27] K. Herndon and T. Meyer. 3d widgets for exploratory scientific visualization. In *Proceedings of UIST '94*, 1994 (to appear).

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

[29] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, 1991.