A Computational Steering Model Applied to Problems in Medicine

Christopher R. Johnson and Steven G. Parker
Department of Computer Science
University of Utah
Salt Lake City, UT 84112.
E-mail: crj@cs.utah.edu and sparker@cs.utah.edu

Abstract

We describe a computational steering model which allows users to interactively change boundary conditions, model geometry, and computational parameters via a graphical user interface. To replace the typical simulation mode – in which the researcher manually sets input parameters, computes results, stores data off to disk, visualizes the results via a separate visualization package, then starts again at the beginning – we have designed software to "close the loop" and allow the visualization to help guide (steer) the design and computation phases of the simulation. We have applied the computational steering model to problems in medicine, specifically to applications in bioelectric field phenomena and biomedical device design.

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.*

Currently, organizing, running and visualizing a new large-scale simulation still requires hours or days. Data I/O and conversion time further complicates and slows the process. Our computational steering model aims to reduce this iteration time (the wall-clock time of the scientist) by reducing both the time required by the scientist for making changes and the computational cost required to effect those changes.

To reduce the time the scientist spends in the current modeling process, we provide an integrated system which will respond to parameter changes automatically. The scientist will be able to interactively change parameters (with a mouse and/or other input devices) and visualize these changes. In addition to being more convenient, this interaction will reduce time wasted by editing mistakes, and will also help reduce computing time by allowing the system to exploit the coherence maintained when only small changes in design parameters are made.

Computational steering has been defined as "the capacity to control the execution of long-running, resource-intensive programs" [1]. 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 (which, for large-scale problems, means dealing with the subtleties of various high-performance architectures). Essentially, the computational steering framework is to engineering as the decathlete is to sports: success is measured by the extent to which many components, often juxtaposed against one another, can work together to bring about an integrated performance.

The construction of a computational steering system has been motivated by our work in applying the computational paradigm to 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, such as (1) interactive programs to construct, manipulate, and display large scale, three-dimensional surface and volumetric meshes [2, 3, 4, 5]; (2) three-dimensional, adaptive, finite element programs for solving elliptic partial differential equations with general boundary conditions and source terms [2, 6, 7, 8]; (3) programs that solve the resulting large system of equations and simultaneously apply global and local regularization to ill-posed inverse problems [2, 9]; and (4) scientific visualization packages for displaying and interpreting the results [3, 4, 5, 10]. 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.

For a computational steering model to be useful, it needs to be applicable to *real world* problems. One of the problems we address, with input from medical device companies, is the application of our steering model to cardiac defibrillation simulation and defibrillation electrode design. While external defibrillation units have been in use for some time, their use is limited by the speed with which a heart attack victim may die from insufficient oxygen to the brain. Lately, research has been initiated to find a way of implanting electrodes within the body to defibrillate a person automatically upon onset of cardiac fibrillation (a common consequence of heart attack). 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. Now that we have produced a model of the human thorax capable of simulating both the endogenous fields of the heart and applied current sources (defibrillation devices), we are able to simulate the multitude of electrode configurations, their sizes, and the magnitudes of defibrillation shocks via computer simulation. Given the large possible number of both external and internal electrode sites, magnitudes, and configurations, however, it is still tedious 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.

To accommodate efficient design and optimization, we have developed (and are continuing to develop) a computational steering environment that allows the user to design bioelectric devices and measure their effectiveness in an interactive graphical environment. Instead of the typical simulation mode – which requires manually setting input parameters, computing results, storing data off to disk, visualizing the results via a separate visualization package, then starting again at the beginning – we have designed software to "close the loop" and allow the visualization to help guide (steer) the design and computation phases of the simulation. The approach we have taken is general. While specifically tailored to bioelectric field problems, the computational infrastructure we are developing should be broadly useful to researchers interested in investigating other science and engineering field problems and design applications.

An Application Problem in Cardiology

Here we address the application of our computational steering model to two problems in electrocardiography, the forward (or direct) ECG problem and simulation of cardiac defibrillation. Mathematically, these problems are governed by the generalized Laplace's equation for electrical conduction in the physical domain Ω [11]:

$$\nabla \cdot \sigma \nabla \Phi = 0 \tag{1}$$

where 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 heart or internal defibrillator electrodes, and Γ_2 is the surface of the torso. 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, which is zero for the insulated boundary.

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

$$\mathbf{J} = -\sigma\nabla\Phi. \qquad (3)$$

The forward ECG problem is characterized by solving equations (1-3) with Φ_0 equal to voltages measured on the heart's surface and yields information regarding the voltages and current flow within the thorax as a function of the endogenous fields of the heart. 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 [12, 13]. 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 our computational steering model 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.

Below we describe a computational steering package we are developing that will allow the user to design electrodes via a CAD type interface, interactively place the electrode(s) directly into the computer model of the thorax using a graphical input device, and automatically change input parameters and boundary conditions as well as the mesh discretization level needed for an accurate finite element solution. The user will then run the simulation. As quickly as the machine allows, the user will be given a graphical representation of the effectiveness of the solution (via calculated current density in the visually represented heart, for example). The user will then decide whether to continue the calculation on more refined meshes for higher accuracy, interactively change the position and/or design of the electrode, or both; or the user may allow the calculation to continue while designing another configuration. While the example given above is targeted to a specific application in computational medicine, the tools created for such a problem have wide application in any engineering field problem and could be modified to other specific governing equations and geometries.

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 cardiology 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 models from MRI (magnetic resonance imaging) scans. A semi-automatic segmentation algorithm is used to classify the relevant tissues [14]. We then utilize two- and three-dimensional Delaunay triangulation algorithms to construct three-dimensional surfaces and/or volumetric meshes [8, 15, 16]. Geometric postprocessing consists of assigning each tetrahedron an electrical conductivity tensor. The range of conductivity represented in the model is approximately 50 to 1 with blood the most conductive and bone the least. The largely parallel orientation of the skeletal muscle results in a higher conductivity along the fibers than across the fiber direction, a fact that necessitated the use of conductivity tensors.

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. body) and anisotropic nature of some of the inhomogeneities (i.e. muscle) in our problem, 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 $N = 25,000 - 1,000,000$ depending on the level of mesh refinement, on the level of interactiveness required by the user, and on the specific goals of the study [2, 3].

Due to the quasi-static nature of the bioelectric field problems we are solving [11], 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[17]. Briefly, suppose we want the accuracy of the finite element approximation to be within a given tolerance, δ of the true solution,

$$||\Phi - \Phi_h|| \leq \delta. \tag{4}$$

We can use an error estimator [18, 7, 8] to refine the mesh so that the resulting error in the finite element approximation of the governing equations decreases to within the limits imposed by equation (4). Such error estimators take the form, for example,

$$\sum_{k \in T_h} (h_k ||\Phi||_{H^2(k)})^2 \leq \frac{\delta^2}{c^2}, \tag{5}$$

where $||\Phi||_{H^2(k)}$ measures the L_2-norm of the partial derivatives of Φ, c is a positive constant, and h_k is a measure of the size of an element. The simulation starts with a coarse discretization level which conforms to the topology of the problem and a given set of boundary conditions. The error estimator is applied to all k elements that make up the finite element mesh, T_h. If the value of the error estimate in a given element exceeds the prescribed tolerance, then the element is refined according to the degree of the error. Obviously, with each iteration of the refinement algorithm, the number of degrees of freedom increases, as does the time necessary to compute subsequent iterations.

Within the computational steering model, 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 computational steering modality, 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 [3, 4, 5, 10]. 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 [19]. For our computational steering model, 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 dataflow 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

Once the geometric model is in place, the computational scientist usually inputs particular boundary conditions, runs the simulation, then pipes the output data to a visualization package to view the results. Upon seeing and interpreting the results, he/she typically then goes back to the input data file, manually changes the input parameters, and runs the simulation again. 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.

To achieve this goal, we have designed a framework based on a dataflow concept to manage data dependencies and computational orderings. In the computational steering framework, the scientist sets input parameters and starts the simulation. The researcher is then able to interact with the model by changing spatial and/or temporal requirements as the solution algorithm progresses. Software systems such as AVS from Application Visualization Systems Inc. [20], Iris Explorer from Silicon Graphics, and Visualization Data Explorer from IBM [21] have made this archetype popular for scientific visualization [22]. Our work has extended this paradigm into the realm of scientific computation.

As a simple example, consider a group of visualization modules within our dataflow network, illustrated in Figure 1. The boxes represent computational algorithms (modules) and the lines represent data pipes between the modules. The topmost box (VectorFieldReader) is responsible for reading a vector field from a file. This vector field may be represented in a number of ways involving a variety of geometric primitives, including regular and unstructured grids. The leftmost box is responsible for calculating streamlines, and the rightmost box is responsible for calculating stream surfaces. The bottom-most box (Salmon) is of central importance to any 3D visualization application. This module provides 3D geometry display and interaction. As the streamline module computes streamlines, it will send geometry (lines or other primitives) to Salmon. Salmon displays the 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.

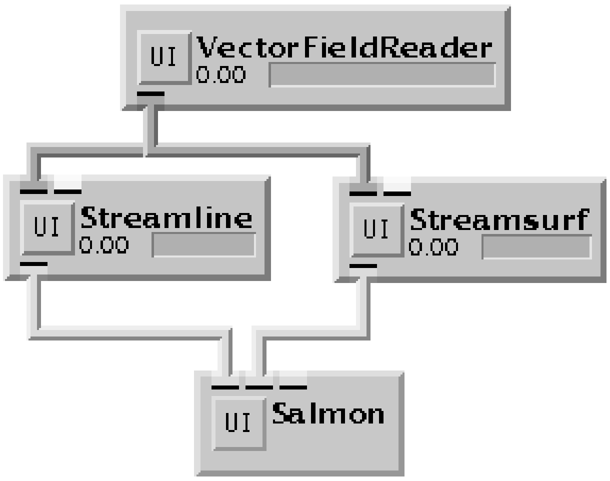

Figure 1: Dataflow Network for a Subset of Visualization Modules

Salmon is also capable of spawning "children" windows in order to display multiple views of the same data. In fact, it was named Salmon because of this feature and it's ability to send messages "upstream" for 3D user interface widgets.

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. The pop-up for the ReadVectorField module allows the user to choose the file from which to read the vector field data structure. The Streamline module's pop-up allows the user to choose integration parameters, and to select between displaying a single streamline or a row of streamlines. The Streamsurf module allows the user to choose surface adaptation parameters, such as it's maximum bending angle. As an example user interface, Figure 2 shows the pop-up windows for the Streamline module.

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. Other changes can be made and the compu-

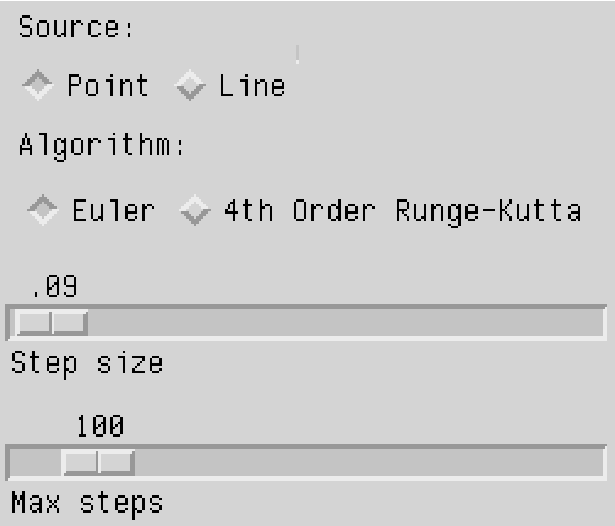

Figure 2: Pop-up Window for the Streamline Module

tations will be cancelled and re-started. In this way, changes can be efficiently made to more than one parameter.

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: A network of modules in the dataflow system 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 Streamline and Streamsurf modules simultaneously. 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* [23, 24] and alternative input devices to assist in the interaction tasks.

Steering a Defibrillator Design Problem

A network (dataflow program) that can be used to implement a computational steering application for cardiac defibrillation is shown in Figure 3. The network 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.

- **GenSurface** will generate two cylindrical electrodes for the defibrillation study. Parameters in the interface allow the scientist to control the discretization of the electrodes.

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

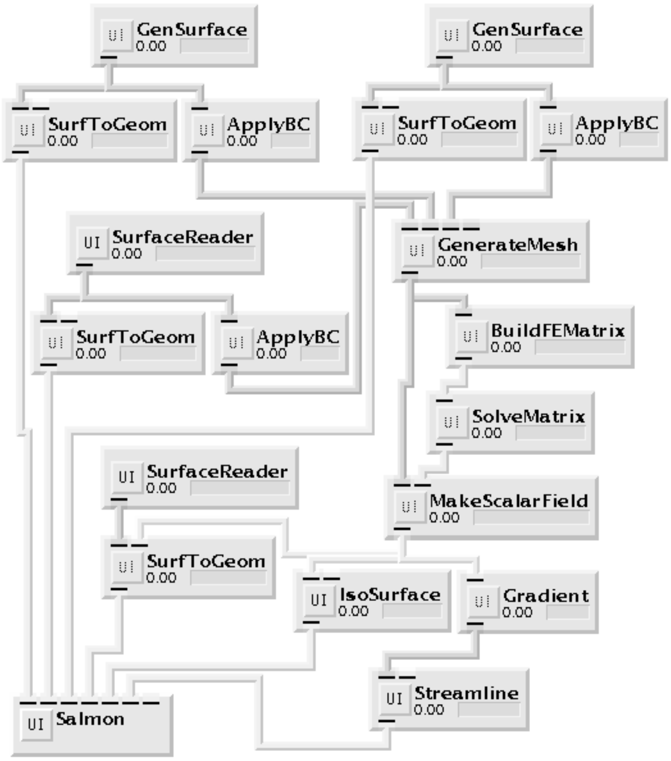

Figure 3: Computational Steering Network for Cardiac Defibrillation

- **ApplyBC** applies boundary conditions to the various surfaces. The torso has a zero-flux Neumann boundary, and the two electrode cylinders have Dirichlet boundary conditions corresponding to their respective voltage. These voltages may be changed interactively.

- **GenerateMesh** discretizes the volume defined on the outside by the torso (the leftmost input) 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 which 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 Laplace's equation, 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 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. 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 [25]. 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.

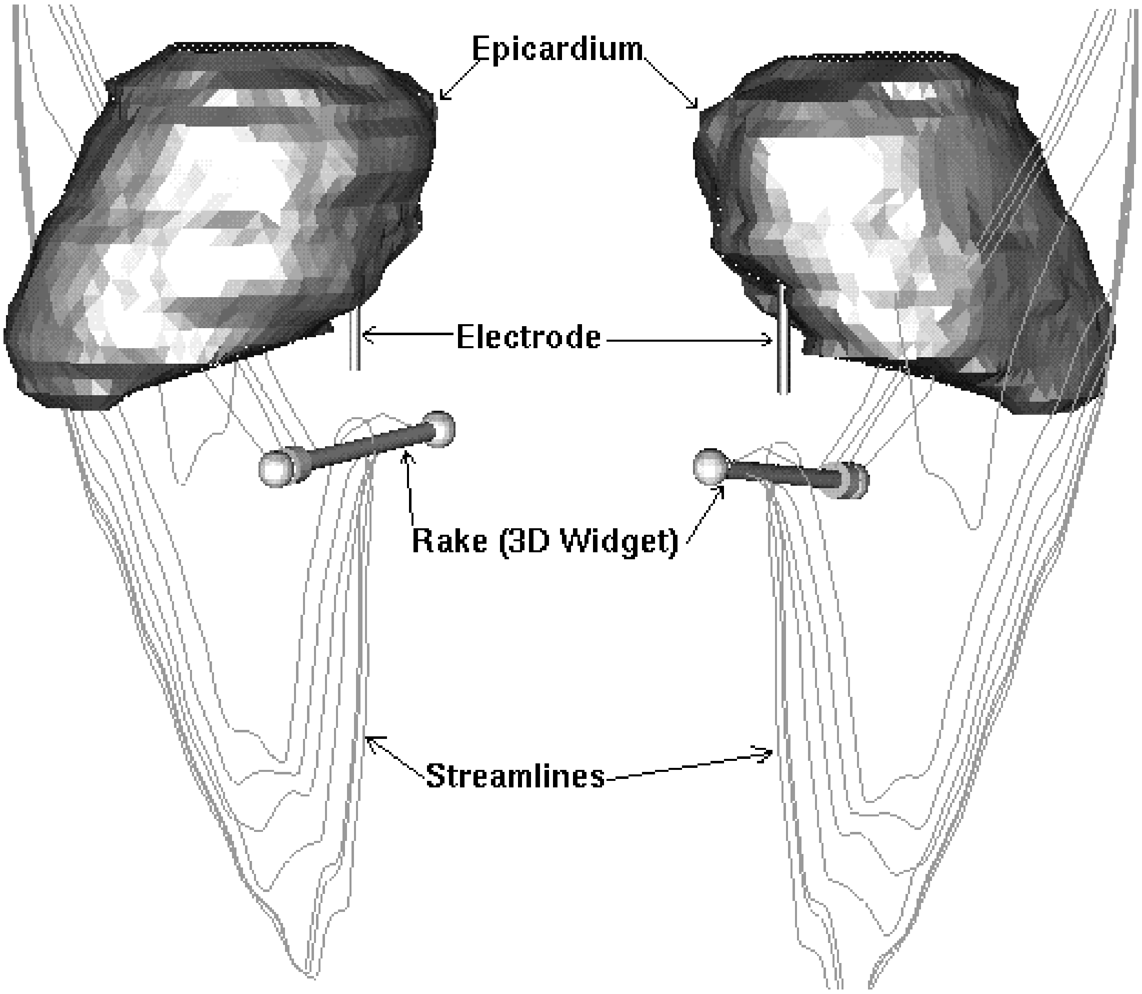

Figure 4: Visualization of a defibrillator design simulation, showing an electrode, the surface of the heart (epicardium), a 3D widget (rake), and electrical current lines (streamlines). The other electrode in the simulation is obscured by the heart. Much of the current leaving the visible electrode travels away from the heart, due to the high conductivity of blood in a nearby vessel.

Each of these modules are simple enough to be managed easily, but when they are joined together, they accomplish a very complex task. A sample visualization from this type of network is shown in Figure 4.

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 [26].

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* [27] 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 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. The meshing algorithm can be jump-started by warping or "morphing" the old mesh to conform to the new geometry. 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 solver will converge rapidly. This concept is similar to exploiting temporal coherence in a time-dependent system by using the previous timestep 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 a computational steering model which 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 machines from Silicon Graphics, using shared memory hardware to communicate between processors in a multi-processor environment. Work is under way to port the system to new architectures, including those without the Motif or OpenGL libraries. In addition, support for distributed memory computers and 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 which this extension will attempt to address.

Acknowledgements

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

References

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

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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).

[9] 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.

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

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

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

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

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

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

[16] 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.

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

[18] P.G. Ciarlet and J.L Lions. *Handbook of Numerical Analysis: Finite Element Methods*, volume 1. North-Holland, Amsterdam, 1991.

[19] H.W. Shen and C.R. Johnson. Differential volume rendering: A fast volume visualization technique for flow animation. In *IEEE Visualization '94*, 1994 (to appear).

[20] 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.

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

[22] 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.

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

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

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

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

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