Interactive Simulation and Visualization

Christopher Johnson, Steven Parker, Charles Hansen,
Gordon Kindlmann, and Yarden Livnat
Center for Scientific Computing and Imaging
Department of Computer Science
University of Utah, Salt Lake City, UT 84112.
E-Mail: crj,sparker,hansen,gk,yarden@cs.utah.edu
Web: http://www.cs.utah.edu/sci

CONTENTS

Abstract

As computational engineering and science applications have grown in size and complexity, the process of analyzing and visualizing the resulting vast amounts of data has become an increasingly difficult task. Traditionally, data analysis and visualization are performed as post-processing steps after a simulation has been run. As simulations have increased in size, this task has become increasingly difficult–often requiring significant computation, high-performance machines, high capacity storage, and high bandwidth networks. *Computational steering* is an emerging technology that addresses this problem by "closing the loop" and providing a mechanism for integrating modeling, simulation, data analysis, and visualization. This integration allows a researcher to interactively control simulations and perform data analysis while avoiding many of the pitfalls associated with the traditional batch/post processing cycle.

In this paper, we describe the application of interactive simulation and visualization as applied to the domain of computational field problems. We discuss the integration of visualization techniques within an integrated problem solving environment, SCIRun, as well as a case study of the simulation and visualization of bioelectric fields.

Keywords

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

I. Simulation Steering and Visualization

Data analysis and visualization play critical roles in the scientific process. Unfortunately, these tasks are often performed only as a post-processing step after batch jobs are run. For this reason, errors invalidating the results of the entire simulation may be discovered only during post-processing. What is more, the decoupling of simulation and analysis/visualization can present serious scientific obstacles to the researcher in interpreting the answers to "what if" questions.

Given the limitations of the batch/post processing cycle, a better approach might be to break the cycle and improve the integration of simulation and visualization. In 1987, the Visualization in Scientific Computing (ViSC) workshop reported as follows [1]:
Scientists not only want to analyze data that results from super-computations; they also want to interpret what is happening to the data during super-computations. Researchers want to *steer* calculations in close-to-real-time; they want to be able to change parameters, resolution or representation, and see the effects. They want to drive the scientific discovery process; they want to *interact* with their data.

Although these thoughts were recorded ten years ago, they express a very simple, contemporary idea: that scientists want more interaction than is permitted by most simulation codes. Computational steering has been defined as "the capacity to control all aspects of the computational science pipeline" [2]. By "computational science pipeline," we mean the succession of steps required to solve computational science and engineering problems. This succession often involves geometric modeling (mesh generation, CAD), simulation (approximation methods, solvers), and visualization (scalar, vector, and tensor fields). We apply the concept of steering 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 to 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 and visualization.

Implementation of an interactive simulation and visualization 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) should

encompasses all of these characteristics, from algorithm development through performance tuning to application steering, for scientific exploration and visualization, and that it should provide a rich environment for accomplishing computational science [3].

In the remainder of this paper, we first describe the application of interactive simulation and visualization as applied to the domain of computational field problems. Within the paper, we discuss the use of interactive visualization techniques within an integrated problem solving environment, SCIRun, and also as applied to the simulation and visualization of bioelectric fields (see the Case Study).

II. Methods for Interactivity

A. Possible Approaches

Interactive visualization typically combines two approaches: providing efficient algorithms for the presentation of data, and providing efficient access to the data.

The first approach is obvious but challenging. Even though computers continually get faster, data sizes are growing at an even more rapid rate. Therefore, the total time from data to picture is not shrinking for many problem domains. Alternative algorithms, such as ray tracing [4] and view dependent algorithms [5] can restore a degree of interactivity for very large datasets. These algorithms each have tradeoffs and are suitable in different scenarios.

The second approach is less obvious but very powerful. Through the integration of visualization tools with simulation codes, a scientist can achieve a new degree of interactivity through the direct visualization and even manipulation of the data. The scientist does not necessarily wait for the computation to finish before interacting with the data, but can interact with a running simulation. While conceptually simple, this approach poses numerous technical challenges.

These two basic approaches raise many important issues. Some of them are explored here.

B. Facets of Interactive Visualization

The goal of interactive visualization is to provide the ability to rapidly examine (and understand) data through visualization methods. Unfortunately, as data sizes grow, standard visualization algorithms do not scale well and different approaches are needed. New algorithms that effectively handle large data must be developed and refined. One common method for visualization is to explore 3D data sets through examination of isosurfaces. To demonstrate the need for algorithmic improvement for visualization, consider isosurface extraction. There are multiple methods for the visualization of isosurfaces, including geometric representation and rendering, direct isosurface rendering and volume rendering. We describe methods for achieving interactive rates with the first method and point out the power of interactive isosurfacing through volume rendering in the accompanying sidebar[1].

B.1 Isosurface Extraction

The standard, non accelerated method for isosurface extraction is the Marching Cubes algorithm [7]. This algorithm checks each cell of the data set to see if it contains an isosurface. If the scientist is exploring the data set, the potential for examining multiple isovalues means that many cells are checked repeatedly when they don't contain data that contributes to the final image.

One can preprocess the data and build specific data structures that allow for rapidly extracting isosurfaces. One such method is the Near Optimal IsoSurface Extraction (NOISE) algorithm [8]. Using a new representation, termed the *span space*, of the underlying domain, the isosurface extraction algorithm is bounded with a worst case complexity of $O(\sqrt{n} + k)$ for the *search phase*, where n is the size of the data set and k is the number of cells in the isosurface. The memory requirement is kept at $O(n)$ while the preprocessing step is $O(n \log n)$. This reduces the search domain for a particular isosurface from all the cells to only those

[1] For examples of direct isosurface rendering, see [4], [6]

cells that *contain* the isosurface [8], [9]. Figure 1 shows a screen dump from an interactive parallel rendering system built with the NOISE algorithm [10]

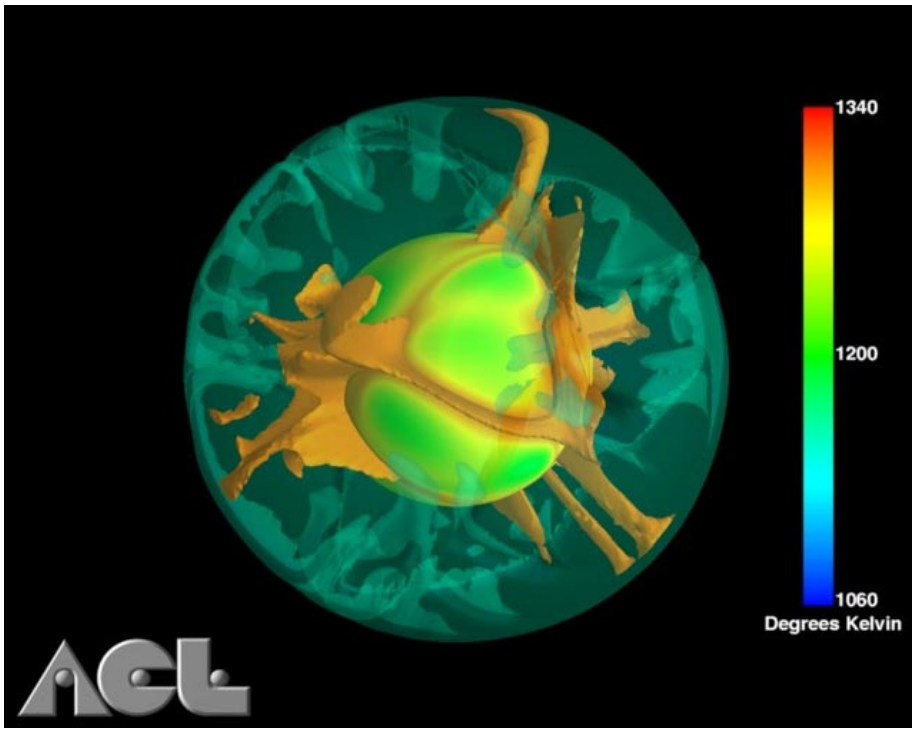


Fig. 1. An earth mantle convection system showing multiple isosurfaces.

B.2 View Dependent Isosurface Extraction

While algorithms such as NOISE have effectively eliminated the search phase bottleneck, the cost of constructing and rendering the isosurface remains high. Many of today's simulation data sets contain very large and complex isosurfaces that can easily overwhelm even state-of-the-art graphics hardware. An output sensitive approach that can further increase interactivity is needed.

Large and complex isosurfaces have two characteristics: 1) many of the polygons that make up the isosurface are smaller than a pixel and 2) the polygons have significant depth complexity, such that the number of polygons that project to a given pixel is quite high. By reducing the required generation of non-visible isosurfaces, we can increase the interactivity for such data sets. Recall that Marching Cubes examined every cell and that search acceleration methods reduced that search to only those cells containing an isosurface. If we can further reduce the search to only those isosurfaces that are *visible* in the final image, we can gain more interactivity. Since many of the polygons project to the same pixel, we can achieve this goal using a view-dependent algorithm.

The WISE algorithm [5], is based on a hierarchical front-to-back traversal of the dataset with dynamic pruning of sections that are hidden from the view point by previously extracted sections of the isosurface. These coarse visibility tests are done in software against a one bit per pixel virtual screen. Finally, the triangulation of the visible cells are forwarded to the graphics accelerator for rendering by the hardware. It is at this stage that the final and exact partial-visibility of the triangles is resolved.

This work explores the middle ground between a mostly hardware based (e.g. Marching Cubes + Z-buffer) algorithm and a purely software (e.g. ray-tracing) algorithm for isosurface extraction. The goal is to reduce the load on the network and/or graphics hardware by performing *some* of the visibility tests in software. The approach leads to an output sensitive method that can reduce the load of other components in the visualization pipeline [22], such as transmission of the isosurface geometry over a network.

View dependent algorithms provide significant improvements for interactivity for very large data. We anticipate significant development using view dependent methods in the coming years.

C. Facets of Computational Steering

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

Such rigidity is especially undesirable given the long life-span of most scientific applications and the wide range of computational requirements found in these applications. Before acceptable interaction and flexibility can be achieved, an attempt to implement steering must address four major facets of the problem: control structures, data distribution, data presentation and user interfaces. These facets may not all be present in every problem, nor do they portray the entire problem - they simply outline fundamental considerations.

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

A computational steering programming "model" is the software architecture used to integrate computational components in a manner that allows the efficient extraction of scientific information and permits changes to simulation parameters and data in a meaningful way. This new architecture often requires some modification of the original scientific code, but the extent and nature of the changes will depend on the model chosen. At one extreme, the scientific program will be completely rewritten and transformed to support steering. Less radical approaches may reuse pieces of the computation or use off-the-shelf visualization packages to simplify the construction of a steerable system [11], [12], [13].

Important to the success of any computational steering system is the means by which it permits the user to specify various types of changes to be made in the simulation. For example, various devices may allow the user to specify changes in the computation, ranging from simple text files to sophisticated scripting languages to graphical user interfaces to three-dimensional widgets. Another less obvious problem is that of integrating changed data into the simulation in a scientifically meaningful fashion. In most coupled systems, it does not make sense to change one quantity without making corresponding changes in other quantities. For example, in a fluid dynamics system, it would not make sense to allow sudden changes in pressure without making corresponding changes in another quantity such as temperature in order to maintain balance under the ideal gas law.

Several approaches can be used to make scientific applications into steerable systems. Each approach has strengths and weaknesses. In many cases, one might use components of all of these approaches.

C.1 Program Instrumentation

One way to implement the steering of an existing scientific application is to make small modifications in the source to provide access points for the parameters and results. This process is called "instrumentation" and typically takes the form of subroutine calls inserted in the code wherever results become available or when new parameters can be used. These calls can transmit data to and from a separate visualization process. They might perform visualization tasks internally, or they might trigger a thread that siphons the data off while the computation continues concurrently. Systems such as Falcon [14], Progress [15] implement this approach. The instrumentation technique has the advantage of being minimally intrusive to an existing scientific code. Instrumentation works well for domain-specific applications and development of new applications

when parameters to be controlled are clearly defined. However, it may provide only limited control over the existing applications as the user may access only the parameters that have been instrumented. This technique also has implementation complications, such as the overhead of data transmission if the data is being sent to a separate visualization process, the stalling of computation if visualization is done internally, and complicated synchronization.

C.2 Directed Scientific Computation

An alternative approach to program instrumentation is to break a code up into various modules that the user controls explicitly by issuing a sequence of commands. One popular approach for doing this is to rely upon scripting languages such as Python or Tcl [16], [17]. This model has been used successfully in commercial packages such as IDL, MATLAB, or Mathematica. It is also being used in large physics applications at both Los Alamos and Lawrence Livermore National Laboratories [11], [18].

The advantage of this method is that it is possible to reuse almost all of the original scientific code. It is also portable and quite easy to implement on most systems since it avoids the problems of managing multiple threads and synchronization. This makes it suitable for controlling most kinds of applications–including large parallel applications on both distributed memory and shared memory systems. The scripting language interface provides expert users with a fine degree of control over most, if not all, program parameters. This system can be also used for scripting long-running simulations, rapid prototyping of new features, and debugging. As an added benefit, most scripting languages provide access to Tk, a toolkit for building graphical user-interfaces. Thus, one could implement a graphical user interface over a command-driven system if needed.

Scripting languages have been used quite successfully in steering a variety of scientific applications [19]. They have also been used in the construction of advanced systems, including SCIRun. They form an ideal control mechanism for the construction of steering systems and other large-scale scientific applications.

C.3 Dedicated Steering Systems

If a scientist is fortunate enough to be designing a new model with computational steering in mind, there is more room for innovation. The SCIRun system [20], [2], [21] has been designed for this scenario. It allows the scientist to construct a simulation using reusable computational components connected within a visual programming environment. Each of these components, as well as the system as a whole, have been designed to integrate modeling, computation, and visualization, and to facilitate the interactive steering of all phases of the simulation.

Unlike directed approaches, the SCIRun system manages each module as an independent thread of execution. This allows the user to interact with the system, even when long-running operations are involved (although data dependencies between operations may force the user to wait for results). The dataflow model used by SCIRun also simplifies synchronization issues, making it relatively easy to “siphon off” data and feed it to various analysis and visualization modules as a simulation progresses.

A dedicated steering system such as SCIRun offers many advanced features, especially if one is writing a new application. While such systems offer a somewhat flexible model that can accommodate existing code, applying them to existing code can be difficult in practice. Scientific applications may not have been written in a manner that is easily translated to such a steering environment. In other cases, there may just be too much “dust on the deck,” in which case it would be easier to start over. Advanced steering systems such as SCIRun may also rely on a style of programming that is difficult to implement on certain machines. For example, the computational model used in SCIRun was designed for use with shared-memory symmetric multiprocessing systems. Implementing the system on a distributed message passing machine involves different challenges, such as synchronization, memory use, and data distribution, among others.

C.4 Data Presentation

Another consideration is the presentation of information to the end user of the steering system. Choices range from off-the-shelf visualization tools to custom analysis and visualization programs [12], [13], [19], [20]. Ideally, a flexible control structure will allow a variety of tools to be mixed and matched for each specific problem.

Often, the full power of a computational steering system comes from the tight integration of scientific codes with visualization tools that were designed for that problem. Visualization is typically used to view the results of the computation, but may be used in other roles as well, such as in visualizing memory usage, algorithm performance, or multiprocessor communication patterns. It may also be used to examine intermediate results, matrix structures, mesh details or domain decompositions.

In other cases, it may be possible to use pre-existing software as a presentation mechanism. For example, a system may use public domain graphing libraries and image processing tools such as gnuplot and xv. Explorer, AVS, and other commercial systems have also been used successfully in steering systems [12].

C.5 Putting it all together

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

Users may also want to write extensions or use the system with their own applications. Unfortunately, doing so may require a detailed understanding of each facet in the problem. Because they perceive the complexity of applying such systems to their own work, many potential users continue to ignore computational steering efforts. While there is no clear answer to this problem, it is clear that this issue will need to be addressed in order for steering to be adopted into mainstream scientific computing efforts.

III. Summary

In recent years, computational steering has started to gain an evolutionary acceptance within, rather than a cause a revolutionary change to, the scientific computing process. This trend is likely to continue, but some factors suggest that the evolution may accelerate:

- The melding of high-end graphics workstations and supercomputers will allow many more scientists to perform both computations and visualizations using the same hardware. Instead of transferring gigabytes of data to a visualization workstation, the researcher will be able to perform the visualization directly on the supercomputer.
- Simulation is playing an increasingly important role in today's scientific and engineering worlds. Increased safety responsibilities, heightened environmental awareness, and cheaper CPU cycles have all increased the motivation for many engineers to do more simulation rather than real-world experiments, which may endanger subjects, use precious–and expensive–resources, cause various kinds of environmental contamination, and be very costly in terms of both money and time.
- Visualization systems are becoming more accessible to average scientists. A few years ago, most scientists did not have ready access to graphics workstations; now, with the advent of PC graphics cards, they are becoming commonplace. Similar changes are in sight for high end 3D graphics engines, which will make fast graphics machines available at a reasonable cost.
- Software development tools are making it easier for scientists to develop and use steering systems.

However, there are barriers to this progress:

- Many supercomputers are currently set up for batch mode processing, not for interactive use. A computational steering system violates many of the assumptions that are made in these batch mode systems. As a

result, site managers and system developers need to recognize the benefits of steering in large-scale applications. A common complaint is that "interactivity" is a waste of CPU cycles even though spending some time interactively setting up a simulation may save tens to hundreds of hours of production time later on.

- Since accounting systems at supercomputer sites focus on counting CPU cycles, performing visualization and rendering tasks using these cycles can be extremely expensive. These costs are highly relevant to the large-scale problems run on large supercomputers, but may not be relevant to the scientist doing medium-sized problems on a desktop superworkstation. There is also room for the investigation of tightly coupled "visualization engines," smaller machines placed at the supercomputer center site that can be used to offload visualization tasks from the primary compute machine.
- The quantity of data generated by large scientific simulations continues to outstrip the capabilities of visualization engines — especially typical desktop workstations. Research into more appropriate visualization algorithms and output sensitive techniques will be necessary.

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

A. Future Directions in Interactive Simulation and Visualization

Interactive simulation and visualization (and interactive steering) will succeed only if such systems can be useful to scientists and engineers. Systems should be modular and easy to extend with existing code. Users of such systems should be able to add new capabilities easily without being overwhelmed in systems programming issues. Steering systems should be adaptable to hardware ranging from the largest of supercomputing systems to low-end workstations and PCs. Finally, steering systems must be demonstrably usable in scientific research. Working prototypes are a start, but we hope that scientists and engineers will soon view steering systems as among the most useful and necessary of their working tools.

Much of our future work will explore the distribution of data and the combination of command-driven techniques with the component-oriented approach used in SCIRun. We are investigating the construction of distributed steering systems involving multiple platforms and languages. Ultimately, we hope that this can lead to the development of steering systems that are highly interactive and easy to extend and modify, that work well with large amounts of data, and that can be used on ordinary workstations, high end servers, and supercomputing systems.

In addition to the software aspects of the computational steering system, the visualization software will continue to evolve. Component oriented designs have been successful for visualization in many different areas (such as the visualization toolkit, *vtk* [22]). However, making such component oriented deigns work for interactive large-scale computational steering systems continues to be a challenge.

Other advances in simulation technology, such as the increased importance of adaptive mesh refinement, present a simultaneous challenge and opportunity for computational steering. While an adaptive mesh is more difficult to manage, it presents a natural vehicle to perform multi-resolution data transmission, visualization, and even to effect user-directed changes in the computation. Adaptive structures will be the focus of much present and future research.

As interactive simulation and visualization systems become more common, issues related to data management will grow critical. It is easy for a researcher to generate hundreds or even thousands of images, datafiles, and results. Organizing that data is a significant problem and much work remains to be done to help in this area. Unfortunately, most existing steering research has focused only on implementation and interactivity. However, as these problems are solved, the emphasis will need to shift towards issues of data management, quality of service, and reproducibility of results.

IV. Acknowledgments

This research was supported in part by awards from the DOE, NSF, and NIH. We also acknowledge facilities from SGI-Utah Visual Supercomputing Center at the University of Utah. The authors would like to thank David Weinstein for his helpful comments and suggestions.

APPENDICES

I. VOLUME VISUALIZATION

Conceptually, direct volume rendering is a simple way to visualize volume data. The individual values in the dataset are made visible by an assignment of optical properties, like color and opacity, which are then composited to form an image. As a tool for scientific visualization, the appeal of direct volume rendering is that in contrast to isosurfacing or segmentation, no intermediate geometric information needs to be calculated, so the process maps from the dataset "directly" to an image.

In practice, a significant amount of work is needed to create an intelligible rendering. The basic problem is creating the mapping from data values to optical properties, called the "transfer function," which plays the critical role of selecting those aspects of the dataset to appear in the rendering. The number of degrees of freedom in transfer functions is huge, but current interfaces for setting them are not constrained by the dataset in question. Thus, it is too easy to find a poor transfer function, and users generally find a good transfer function only after a slow process of trial and error.

Our work on this problem has focused on a specific use of direct volume rendering, visualizing scanned medical data (such as CT or MRI) to display the surfaces of organs or bone. While showing these boundaries may seem to be a problem of edge detection (a problem, in other words, of computer vision), the two problems differ in a subtle but important way. While edge detection seeks to locate edges within the two or three spatial dimensions of the image, we need to "locate" boundaries within the range of data values occurring in the dataset, values which represent some physical property like radio-opacity or proton density. It is the lack of a spatial component to this process which makes it unintuitive.

Our solution is to create the transfer function in two distinct steps. The first step is an automated analysis using metrics borrowed from edge detection, but which are then projected into the space of data values. The information accumulated this way is used to compute a "distance function," which gives the signed distance to an object boundary as a function of data value. Having the distance function simplifies the user's task immensely, because it can constrain the otherwise unwieldy parameter space of transfer functions to only those that emphasize the boundaries within the dataset. In the second step, the user generates a "boundary appearance function," which maps distance (rather than data value) to color and opacity.

Mathematically, the transfer function is created as the composition of the distance function and the boundary appearance function. This combines the two steps, the automatically measured information about the boundaries, and the user-specified information about their desired appearance (such as thickness, sharpness, translucency, and color). Setting a transfer function is now simpler, because the hard part of the job has been done already. The user retains control over the transfer function, but at a comfortable remove from the underlying space of raw data values.

This two-step approach has immediate relevance to more complex rendering tasks. The domain of the transfer function may not be just the one-dimensional range of data values, as described here, but some higher-dimensional feature space. However, as long as a distance function can be computed for this new domain, the user's same boundary appearance function can used to create the transfer function. One example of such a domain is the two-dimensional space of data value and gradient magnitude (a first derivative). Figure 2 shows volume renderings from the visible woman data set [23] made with "two-dimensional" opacity functions, shown here as the inset gray-scale images. The horizontal and vertical axes represent data value and gradient magnitude respectively, and brightness represents opacity. If the initially generated opacity function shows more features than desired (upper left), the structure of the opacity function makes it simple to select individual boundaries for rendering.

Abstracted levels of interaction such as this become more important as the size of datasets, and hence required rendering time, grow with advances in measurement equipment and techniques. Also, where datasets and associated volume rendering methods are more complex (such as in vector or tensor visualization), methods for guiding the user towards useful parameter settings, based on information about the goal of the visualization, become a necessary part of generating informative scientific visualizations. Research in these areas is currently underway.

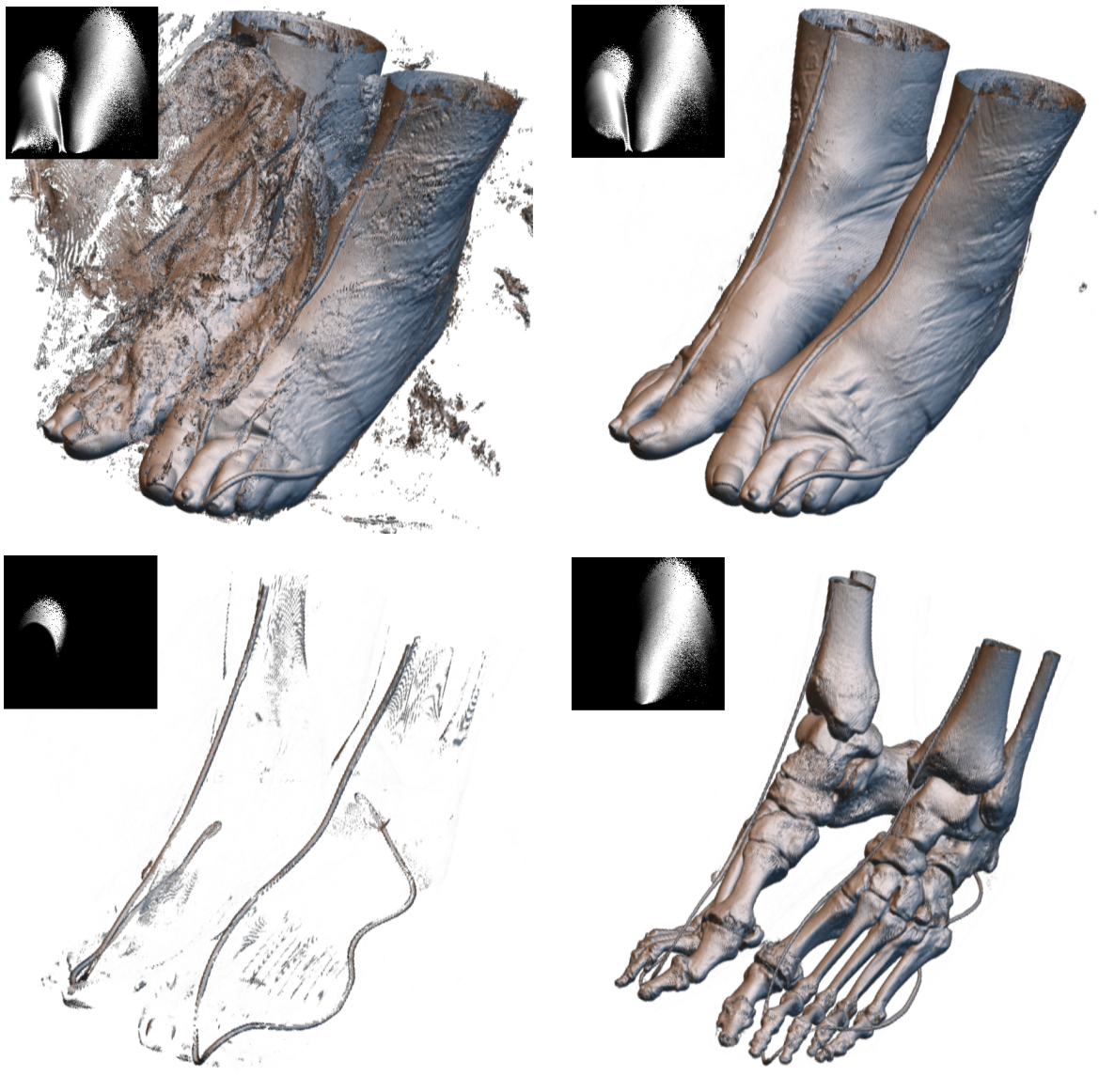


Fig. 2. Manipulation of an automatically generated two-dimensional opacity function to selectively render different material boundaries: skin (upper right), bone (lower right), and the registration cord laced around the body prior to scanning (lower left).

II. Case Study: Computational Bioelectric Fields

Much of the problem solving environment research at Utah has focused on SCIRun, an interactive environment for creating and steering scientific applications [2], [24], [20], [3]. SCIRun is a scientific programming environment that allows the interactive construction and steering of large-scale scientific computations. A scientific application is constructed by connecting computational elements (modules) to form a program (network). This program may contain several computational elements as well as several visualization elements, all of which work together in orchestrating a solution to a scientific problem. Geometric inputs and computational parameters may be changed interactively, and the results of these changes provide immediate feedback to the investigator.

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

Every year, approximately 500,000 people die suddenly because of abnormalities in their hearts' electrical systems (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 available to researchers, most past design work on defibrillation devices has relied on animal studies. In order to provide an alternative based on human geometry, we have constructed a large scale computer model of the human thorax, the Utah Torso Model [25], [26], for simulating both the endogenous fields of the heart and applied current sources (defibrillation devices). Using this model, we are also able to simulate a 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 using electroencephalograms, or EEGs, to measure changes in the patterns of the scalp's electrical activity, physicians can detect some forms of neurological disorders. However, these measurements provide physicians with only a blurred projection of brain activity. A pervasive problem in neuroscience is determining which regions of the brain are active, given voltage measurements at the scalp. If accurate solutions to such problems could be obtained, neurologists would gain non-invasive access to patient-specific cortical activity. Access to such data would ultimately increase the number of patients who could be effectively treated for neural pathologies such as multi-focal epilepsy.

To solve these two bioelectric field problems in medicine, we are using SCIRun. In the first case to design internal defibrillator devices and measure their effectiveness in an interactive graphical environment. Similarly, in the second case, we are using SCIRun to develop and test computational models of epilepsy and localize the focus of electrical activity within the brain due to an epileptic seizure.

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 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.

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 within a detailed model of the human thorax.

One of the challenges of solving the inverse EEG source localization problem is choosing an initial configuration for the search method, such as a downhill simplex method, used to localize the electrical sources. A good choice can result in rapid convergence, whereas a bad choice can cause the algorithm to search randomly for a very long time before closing in on the solution. Furthermore, because the solution space has many local minima, it is necessary to re-seed the algorithm many times in order to find the global minimum.

We have brought the user into the loop by enabling seed-point selection within the model. The user can now seed specifically within physiologically plausible regions. This focus enables the algorithm to converge much more quickly, rather than getting lost in non-global, local minima.

An image of our algorithm running within the SCIRun environment is shown in Figure 4. Interacting with the model depicted in the OpenGL rendering window, the user can seed the downhill simplex search algorithm with sources placed in physiologically plausible regions of the model. In practice, this computational steering capability translates into more than a 50% reduction in the number of iterations required for simplex convergence.

Even simulated experiments of such complexity can be time- and cost-prohibitive. However, using SCIRun, an engineer can interactively steer such a simulation. An engineer can try an electrode configuration and start

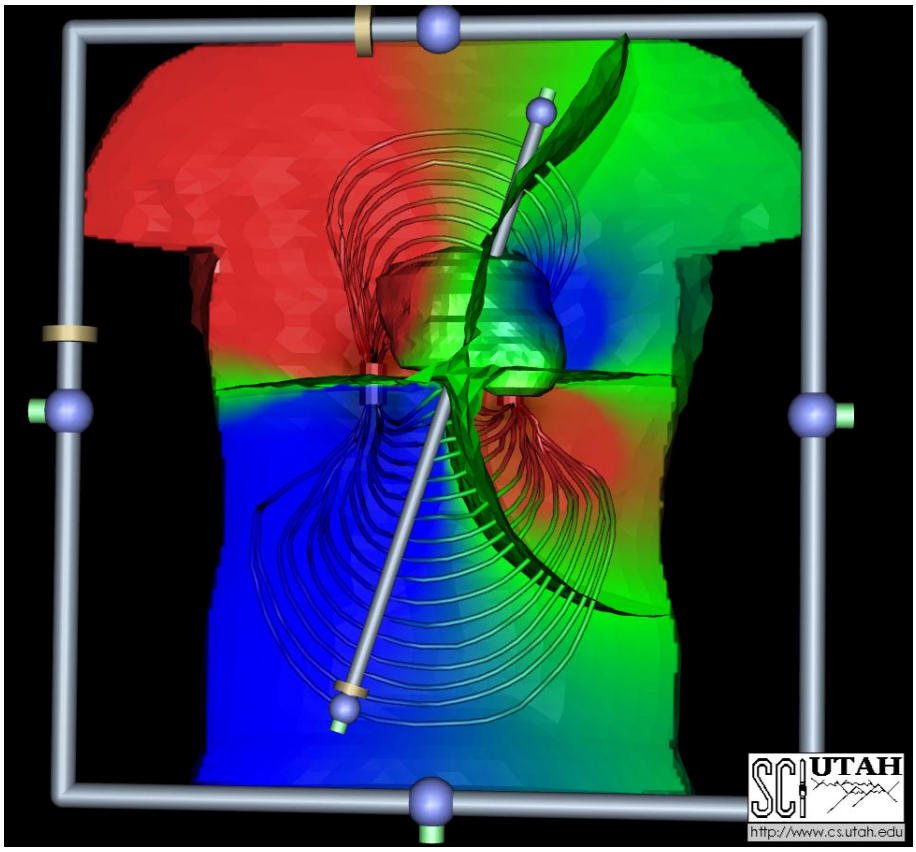
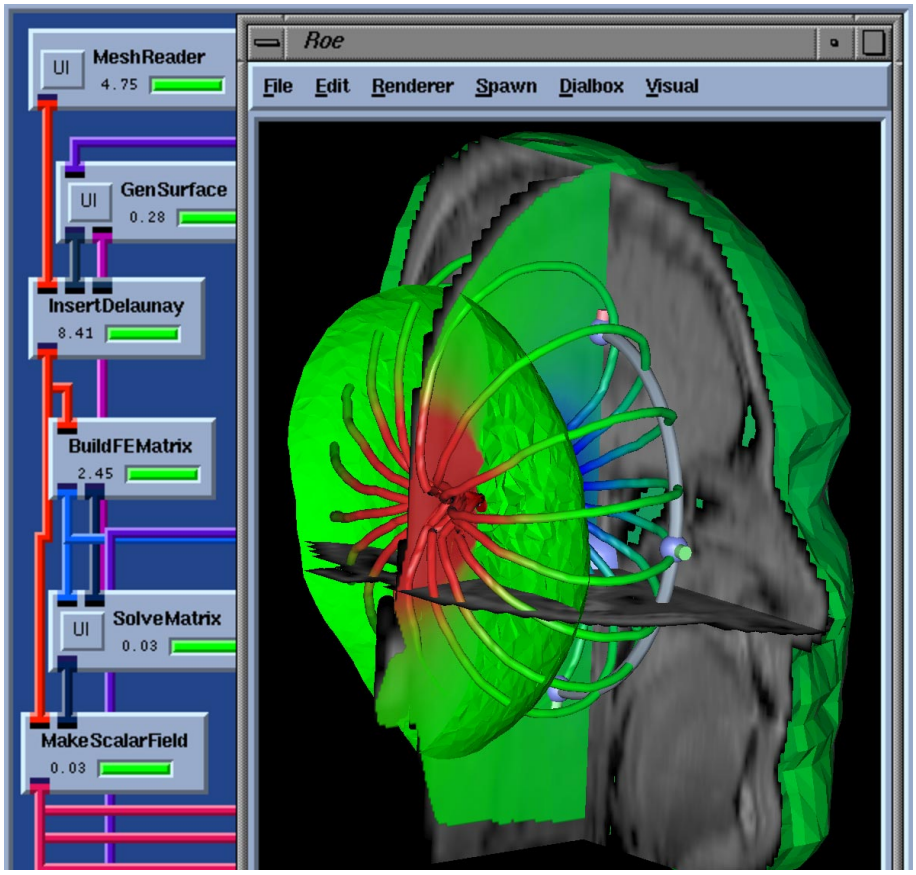


Fig. 3. 3D adaptive finite element solution of an internal defibrillation simulation. A user can interactively investigate the isovoltage surfaces and the lines of electric current flow, as well as change the position and shape of the electrode source.

the simulation. As the simulation progresses, the engineer can view the intermediate results. Long before the system completes a detailed solution, the engineer might determine that the configuration is not acceptable, and might therefore try a new configuration and restart the simulation. Instead of throwing everything away and starting over, SCIRun uses temporal and spatial coherence to compute only those aspects that have changed between the previous simulation(s) and the new simulation. In scenarios where only small changes are made to input parameters, a significant CPU and storage savings can be achieved.

There are many engineering design problems that would benefit from such a system, ranging from the biomedical problems discussed here, to traditional mechanical design, to CFD and computational combustion design. SCIRun is forming the core of two different Problem Solving Environments oriented towards such problems. The first is the Uintah Problem Solving Environment, targeted at large-scale computational combustion problems within the Center for Simulation of Accidental Fires at Explosions (C-SAFE), a DOE ASCI ASAP center [27]. The second environment is BioPSE, a research tool for bioelectric fields, being developed by the NIH Center for Bioelectric Field Modeling, Simulation, and Visualization at the University of Utah. Engineers in these fields desire a tool that will enable them to easily experiment with new geometric configurations and also to modify computational models and numerical algorithms to achieve more accurate results or compute the solution more efficiently. As an integrated visualization environment combined with large-scale computational capabilities, SCIRun is a powerful tool in engineering design.


UI
MeshReader
4.75
UI
GenSurface
0.28
InsertDelaunay
8.41
BuildFEMatrix
2.45
UI
SolveMatrix
0.03
MakeScalarField
0.03
Roe
File
Edit
Renderer
Spawn
Dialbox
Visual