SCIRun: A Scientific Programming Environment for Computational Steering

Steven G. Parker
sparker@cs.utah.edu
http://www.cs.utah.edu/~sparker
(801)581-8224

Christopher R. Johnson
crj@cs.utah.edu
http://www.cs.utah.edu/~crj

Department of Computer Science
3190 Mechanical Engineering Bldg.
University of Utah
SLC, Utah 84112

© Copyright 1995 by ACM, Inc.

Keywords

steering, interactive, dataflow, visual programming, scientific computing

Abstract

We present the design, implementation and application of SCIRun, a scientific programming environment that allows the interactive construction, debugging and steering of large scale scientific computations. Using this "computational workbench," a scientist can design and modify simulations interactively via a dataflow programming model. SCIRun enables scientists to design and modify models and automatically change parameters and boundary conditions as well as the mesh discretization level needed for an accurate numerical solution. 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. To make the dataflow programming paradigm applicable to large scientific problems, we have identified ways to avoid the excessive memory use inherent in standard dataflow implementations, and have implemented

fine-grained dataflow in order to further promote computational efficiency. In this paper, we describe applications of the SCIRun system to several problems in computational medicine. In addition, an we have included an interactive demo program in the form of an application of SCIRun system to a small electrostatic field problem.

Body of Paper

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. Tuning the simulation is accomplished in part by using SCIRun's self instrumentation capabilities.

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.

Towards a Comprehensive Development Environment for Scientific Computing

The typical process of constructing a computational model consists of the following steps:

1. Create and/or modify a discretized geometric model
2. Create and/or modify initial conditions and/or boundary conditions
3. Compute numerical approximations to the governing equation(s), storing results on disk
4. Visualize and/or analyze results using a separate visualization package
5. Make appropriate changes to the model
6. Go back to step 1, 2 and/or 3.

The "art" of obtaining valuable results from a model has up until now required a scientist to execute this process time and time again. Changes made to the model, input parameters, or computational processes are typically made using rudimentary tools (text editors being the most common). Although the experienced scientist will instill some degree of automation, the process is still time consuming and inefficient.

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.

Difficulties in creating such a program arise from the need to integrate a wide range of disparate computing disciplines (such as user interface technology, 3D graphics, parallel computing, programming languages, and numerical analysis) with a wide range of equally disparate application disciplines (such as medicine, meteorology, fluid dynamics, geology, physics, and chemistry). Our approach to overcoming these difficulties is to separate the components of the problem. SCIRun's dataflow model employs "modules" that can be tailored for each application or computing discipline. Although this method is proving successful at partitioning many of the complexities, we have found that some complexities remain, such as the burdens of parallel computing and user interfaces. Much work goes into simplifying the programming interfaces to these features so that they will be used, rather than ignored, by module implementors.

Steering

The primary purpose of SCIRun is to enable the user to interactively control scientific simulations while the computation is in progress [1]. This control allows the user to vary boundary conditions, model geometries, or various computational parameters during simulation. Currently, many debugging systems provide this capability in a very raw, low-level form. SCIRun is designed to provide high-level control over parameters in an efficient and intuitive way, through graphical user interfaces and scientific visualization. These methods permit the scientist or engineer to "close the loop" and use the visualization to steer phases of the computation.

The ability to steer a large scale simulation provides many advantages to the scientific programmer. As changes in parameters become more instantaneous, the cause-effect relationships within the simulation become more evident, allowing the scientist to develop more intuition about the effect of problem parameters, to detect program bugs,

to develop insight into the operation of an algorithm, or to deepen an understanding of the physics of the problem(s) being studied.

The scientific investigation process relies heavily on answers to a range of "What if?" questions. Computational steering allows these questions to be answered more efficiently and therefore to guide the investigation as it occurs.

Overview

The Dataflow Programming Model

The dataflow programming paradigm has proven useful in many applications. In the scientific community, it has been successfully applied in several scientific visualization packages [2-6], including AVS from Advanced Visual Systems Inc., and Iris Explorer from SGI.

Definitions

The following terms will be used to represent the components of our dataflow program.

- A **module**, drawn as a box in the network, represents an algorithm or operation. A set of input ports and a set of output ports define its parameters.

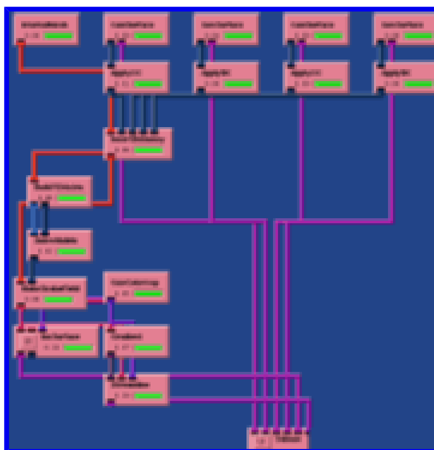
Figure 1. Icon of a module

- A **port** provides a connecting point for routing data to different stages of the computation. Ports are typed: each datatype has a different color, and datatypes cannot be mixed. In SCIRun, ports can be added to and removed from a module dynamically.
- A **datatype** represents a concept behind the numbers. Datatypes are quantities such as scalar fields or matrices, but are not the actual representations, such as regular grids, unstructured grids, banded matrices, sparse matrices, etc.
- A **connection** connects two modules together: the output port of one module is connected to the input port of another module. These connections control where the data is sent to be computed. Output ports can be connected to multiple input

ports, but input ports accept only a single connection.

- A **network** consists of a set of modules and the connections between them. This represents a complete dataflow "program."

Figure 2. An example network



Efficiency in dataflow implementations

While these dataflow systems are useful for prototyping visualizations, many users have found them to be too resource intensive to use in solving large scale problems. In order to extend the usefulness of such a paradigm to computation on large datasets, we must address several efficiency issues. Fortunately, most of the inefficiencies in current systems are not inherent to the dataflow model, but are peculiar to implementations of the concept.

Excessive memory usage is one of the main sources of inefficiencies in current systems. In order to keep the memory use within acceptable limits, we avoid making multiple copies of datasets and use fine-grained dataflow where appropriate.

To keep or not to keep

In order to provide interactive response on small datasets, most dataflow systems maintain a copy of the dataset at each stage in the dataflow pipeline. This is not always desirable. Sometimes it may be more efficient to recompute intermediate datasets than to store them. This is especially true when datasets are large and when maintaining extra copies of the dataset will cause to machine to page to disk.

Even when a system does not maintain several copies of the same numbers, copying can become a problem. The dataset may be transformed several times in the different stages of a dataflow network, but still represent at least a subset of the original data. For example, when a dataset is converted to geometrical objects for display, the data is

transformed (to coordinates of the geometrical objects) and copied. With large datasets, making even one copy can increase memory usage of the program to a point where excessive paging will destroy the performance of the machine.

Deciding which datasets need to be stored and which ones should be recomputed is not a straightforward task. Currently, SCIRun delegates this decision to the user, by allowing him/her to interactively select which datasets are to be retained. By default, all intermediate datasets are retained, but graphical switches at each storage point can be used to discard these intermediate results. Heuristics are being investigated to make this decision more automatic.

Handles

Another method of reducing dataset copying is by sharing pointers to data wherever possible. SCIRun controls deallocation of these shared data structures by maintaining reference counts to the dataflow data. In the graphical user interface, these counts are invisible to the user. When programming a new module in C++, SCIRun maintains these reference counts automatically through the use of handles. Handles are simple C++ classes which maintain reference counts in an object when the handle is created, copied and destroyed. The last handle to reference an object will destroy the object and release resources.

Fine-grained/Coarse-grained dataflow

Many operations do not operate on the entire dataset at once. For example, Song and Golin [7] showed that the marching cubes [8] iso-surfacing algorithm could be made 17% more efficient through the use of fine grained communication of the data set. On a multiprocessor machine, the fine-grained dataflow model was 65-76% more efficient than the standard coarse-grained dataflow implementation.

These improvements show the benefits of using fine-grained dataflow in some applications. However, it is difficult to realize fine-grained dataflow while maintaining the simplicity and flexibility of the modules' code. In order to overcome this difficulty, SCIRun uses several different protocols for each datatype. Each protocol defines a different ordering of the data. Modules can promise to access the data in a predefined order, and if upstream modules can deliver the data in the negotiated order, then a significant increase in efficiency can be observed. When portions of the data is no longer needed, the memory can be released. The programming interface to the dataset does not change, making it much simpler for modules to "speak" several of these protocols.

More efficiency

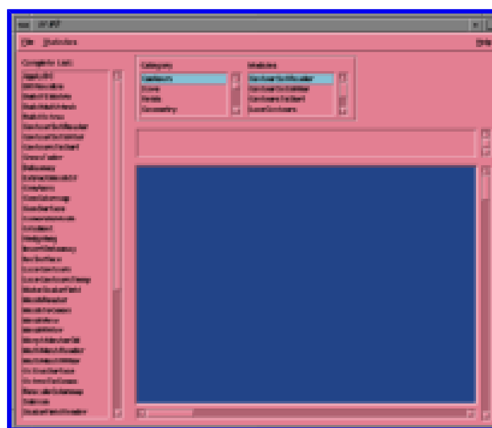
Many changes made to the parameters of a simulation are small and incremental. The SCIRun system attempts to exploit problem coherence to make computing the effects of these changes more efficient. For example, iterative matrix solvers use the previous solution as an initial guess for computing the new solution. If changes to the model are small, the solver will converge much more rapidly. Even if the parameter changes are large, a guess based on a previous solution is seldom worse than any other random guess, such as the zero vector which is typically used as an initial guess.

Other examples of exploiting coherence include limiting regridding of a finite element mesh to changed regions, reusing unchanged portions of a matrix, or avoiding the recomputation of mesh connectivities. The traditional scientific computation model does not facilitate the convenient exploitation of these types of coherence.

Implementation

SCIRun is currently implemented in C++, using TCL/TK for a user interface and scripting language, and OpenGL for 3D graphics. SCIRun is multithreaded: each module can run on a separate processor in a multiprocessor system. In addition, each module can be parallelized internally to take maximum advantage of multiprocessor resources. The multithreaded design allows SCIRun to respond to user input, even while lengthy computations are in progress.

Figure 3. The SCIRun main window



Input/Output

SCIRun employs a simple persistent object system to store data on disk. The data file

completely identifies its type (including version numbers) to subsequent executions of SCIRun. This format drastically reduces data file confusion and the need for complex file naming schemes. The data files encapsulate inheritance to capture the exact representation of a datatype and provide support for both ASCII and portable binary input/output.

User interfaces

Many user interface components (widgets) that are in common use today are not adequate for scientific applications, but are intended for integer or fixed-point data. 2D Widgets are being developed that can be used to specify numbers in scientific notation, thereby allowing for a semi-infinite range and accommodating the specification of several significant figures. Many scientists also need to be able to type in numbers directly. This should also be possible in the user interface.

Many of these needs have been addressed in a prototype version of the "exponential scale" user interface component. The exponential scale allows the user to specify numbers in scientific notation, and allows complete control over all scientific notation. The "+" and "-" buttons allow the user to select the sensitivity of the slider, providing a varying granularity of control.

Figure 4. Exponential Scale Widget Prototype

3D Widgets

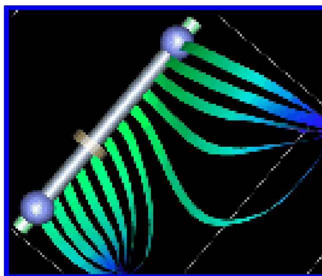
In addition to traditional mouse-based user interfaces, SCIRun relies heavily on "3D Widgets" [9-12] for complex 3D user interaction. Traditional 2D user interfaces have proven inadequate for specifying positions, shapes or other parameters in three-dimensional spaces. A 3D widget consists of extra objects which are placed in views of the visualization, and mouse interaction with these objects can provide complete control over many 3D parameters. As an example of a 3D widget, consider the "rake", composed of cylinders and spheres, which is used by the streamline module to specify an array of starting points for particle advection. These starting points are defined by a line segment in space. The position of these endpoints are more accurately marked by the spherical ends than by numbers on a slider or in a text file. Likewise, manipulating these endpoints can be more accurately performed using the 3D objects of the 3D widget. Integration of 3D widget objects with other objects in the scene scene

can help the user to understand the location of the 3D widget within the dataset.

The widgets can be interactively moved with a 2D mouse, but when coupled with camera rotations, the user can move it to any point in 3D space. Different parts of the widget can perform slightly different tasks. In the rake example shown below, the green end cylinders provide control over the length of the rake, without changing position or orientation. The blue balls on the end of the rake can re-position and re-orient the rake without changing its length. The gray cylinder in the center can be used to re-position the rake, without changing its orientation or length. Finally, the brown cylinder in the middle, is a slider that provides control over the density of the streamlines.

We have developed several widgets that specify points, lines, rectangles, circles and boxes in space. In addition, we are working on several widgets to assist in the creation and manipulation of scenes and animations. The widgets are designed in a consistent way so that a user may develop intuition for their operation.

Figure 5. The rake - an application of 3D widgets in Scientific Visualization

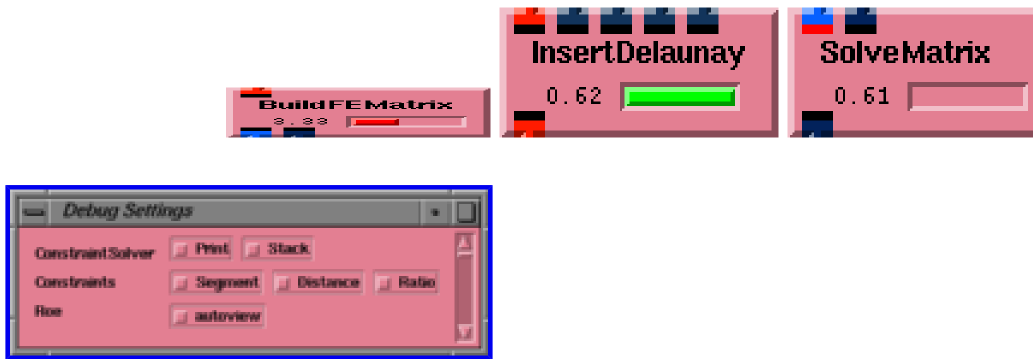


Self instrumentation

SCIRun provides several built-in tools to help tune and debug SCIRun components. The currently implemented tools are described and shown below. Other performance and debugging tools currently under development include tools to visualize parallel and distributed computing resources.

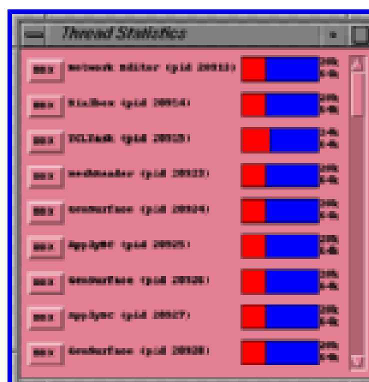
- Progress Meters - Show the fraction of completion of a particular module and the CPU time currently used by the module. The progress meters provide a visual cue for CPU run times of single components of a computation. They are particularly useful for identifying computational bottlenecks. The following three module icons display the three different states of this feature. From left to right, they are Waiting for Data, In Progress, and Completed.

Figure 6. Progress graph states



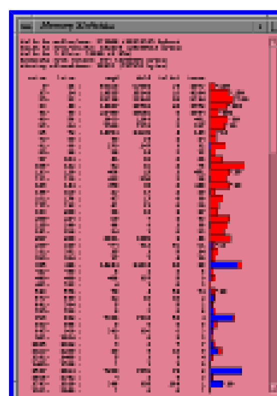
- Thread Display - Shows each active thread in the system, with the size of stack allocated to the thread as well as the amount of the stack that has actually been used. In addition, a button is provided for each thread that will attach the thread to the user's favorite debugger.

Figure 8. Thread status window



- Memory Usage Statistics - Displays a count of the numbers of "chunks" of memory requested and deleted for several different size categories. This output is displayed both graphically and numerically for both quick reference and detailed analysis. The statistics are updated twice per second. This tool can be used to help track down memory leaks.

Figure 9. Memory statistics display



Some of the Datatypes

As a **scientific** programming environment, SCIRun has many different datatypes that represent scientific data. These datatypes include:

- **Mesh** - Unstructured grids consisting of a set of Nodes (datapoints) and a set of Elements (Connectivities). The mesh datatype is used in finite element simulations to define an interpolation in space. When combined with finite element solutions, the mesh data structure can represent Scalar and Vector fields.
- **Surface** - A boundary delimiter in finite element mesh generation. Currently surfaces can be represented as exact cylinders, spheres, or as vertex/triangle meshes. Other representations are easily added. Surfaces can also be used in conjunction with scalar field visualization to view potential maps on any surface in the domain.
- **Matrix** - A matrix typically used to represent equations of the form **Ax=b**, where **A** is an **n** by **m** matrix, **x** is an **m** by **j** matrix, and **b** is an **j** by **n** matrix. The matrix can currently be represented in two forms - as a dense matrix, and as a sparse matrix stored in a compressed row format. Note that the use of this matrix is much more concrete than the the use of an array in typical scientific applications. The matrix datatype is intended to represent only sets of simultaneous linear equations.
- **ScalarField** - A scalar function of space. Currently the scalar field can be represented using a regular grid which is trilinearly interpolated, or as a 3D unstructured mesh (using the **Mesh** data structure described above). Other representations such as symbolically represented fields have been identified as useful, but are not currently implemented.
- **VectorField** - A vector function of space. Similar in function and representation to the **ScalarField** structure described above.
- **Geometry** - Geometry is used by visualization modules, 3D user interfaces and other modules to create objects for display in the graphical viewer. The list constantly grows, but currently implemented objects include:
  - Cone
  - Cylinder
  - Disc
  - Line/Polyline
  - Point
  - Sphere
  - Tetra
  - Torus

  - Triangle/TriangleStrip
  - Tube (Cylinder of varying radius along a path)

  Geometry objects also include several object types for composition, display control, material properties, and manipulation.

In a SCIRun dataflow network, different representations of these datatypes are indistinguishable. This is so that the casual user will find consistent interfaces to all modules. For example, the same isosurface module is used for both unstructured grids and regular grids, even though the two isosurfacing algorithms are very different. A programmer writing the C++ code for a module may have different needs. Whenever possible, then, the actual representation of the data is hidden from the programmer, but this is not always desirable for the sake of performance. For example, a programmer writing the C++ code for a module may have different needs. In these situations, the programmer may query the exact representation of the datatype in order to execute the most optimal algorithm.

In many cases, choosing the appropriate algorithm for the data is a simple task, as when one is building a finite element matrix from an structured mesh as opposed to from an unstructured mesh. However, many choices are not as simple, such as deciding whether to use a structured mesh or an unstructured mesh to model a specific problem.

SCIRun does **NOT** attempt to be a "black box" problem solver that somehow manages to make all of the right choices. However, SCIRun provides a sound set of default values and a small collection of heuristics to make many choices automatically. For example, a finite-element matrix may know that a sparse matrix is going to be more efficient than a dense matrix for a particular problem, so it will use the sparse matrix by default. If the user knows more about the problem, he/she may wish to use a dense matrix. Ideally, the user should never be limited from making these choices, but the program might help discourage the user from making the wrong choice.

Some of the Modules

There is an increasing number of modules available in SCIRun. Some modules are created to solve a particular problem, but most of them are general-purpose computational elements that can be used in a variety of contexts. Several of these general purpose modules are described here.

- **BuildFEMatrix** - Composes a matrix that computes a finite element approximation of the governing equation over the input domain. Tags on the input mesh allow the integration of various boundary conditions.

- **SolveMatrix** - Computes the solution to a matrix equation using one of several methods, such as preconditioned conjugate gradient iteration, Gaussian elimination, or LU decomposition.
- **Delaunay** - Constructs a tetrahedral mesh from a set of points, using the Delaunay criteria.
- **InsertDelaunay** - Inserts new points into an existing tetrahedral mesh. These points are inserted according the Delaunay criteria. This is useful for changing boundaries or sources in a mostly static mesh.
- **Salmon** - Displays 3D objects from various sources in a window. Salmon is responsible for interaction with 3D widgets, for providing multiple views of the geometry and for manipulating these views. Salmon is a key component in most dataflow programs.
- **MeshView** - An application module for interactively visualizing large scale 3D unstructured finite element meshes. The program includes two real-time interactive methods that overcome the difficulties of visualizing unstructured 3D grids on 2D screens: a clipping "surface" utility and a growth algorithm. Used separately or together, these utilities allow the user to better explore complicated unstructured meshes.

Figure 10. The MeshView interface

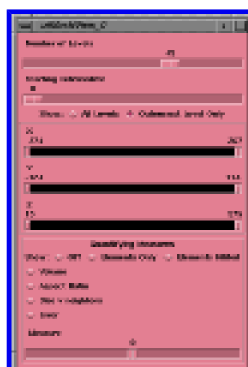
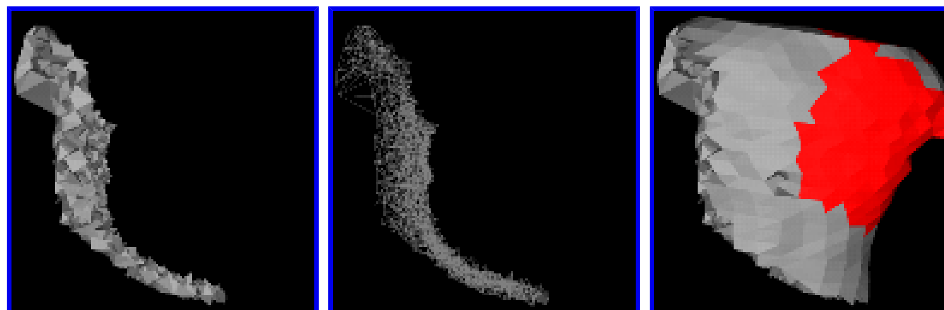
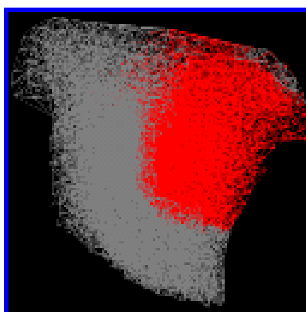


Figure 11. Several views of a mesh of the Utah Torso model





Applications

SCIRun is currently being applied to several problems in computational medicine, for both visualization and computation.

Vector based automatic segmentation

Given a volumetric data set (in this case MR data), it is often desirable to segment the data by tissue (material) type. The method employed here is an extension of the grey-scale range segmentation method, where each voxel is classified according to its scalar grey-scale value (generally measured from 0-255). Where the original method uses only a single grey-scale value for each voxel, we extend the segmentation technique to include an n-dimensional vector for each voxel. This technique requires multiple volume data sets as input, all with identical spatial domains. Ideally, each volume would show different properties of the material, and the resultant vectors could be easily segmented into the correct material types. In our example with MRI scans, the patient underwent two scans with different "weightings." An MRI weighting specifies the frequency of the magnetic pulse, the duration of the pulse, and the relaxation time between pulses. These parameters determine the "material to intensity mapping" that will result from the scan. For this example, we have two volumes, each containing 81 256 x 256 MRI slices from the same patient.

Figure 12. Two MRI images showing the different data obtained for two different weightings. The left image is a T1 weighting, and the right image is a T2 weighting.

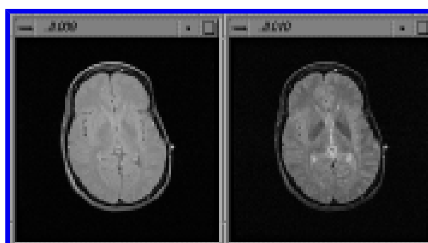
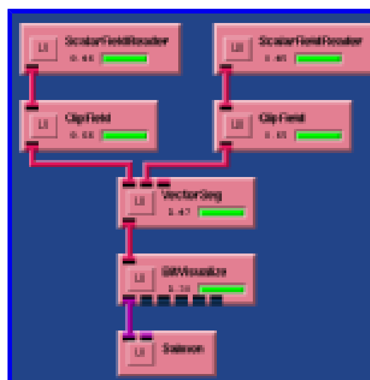


Figure 13. The network used for vector based automatic segmentation



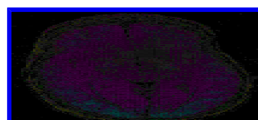
Two seperate scalar fields are read from files using ScalarFieldReader's. These fields are then clipped using two ClipField modules. The ClipField modules are linked so that they will always define the same volume (i.e. if you move one volume's clipping planes, the other's move as well). If these modules were not included, segmenting the volume would take a long time, and the process of determining the best material ranges would not be nearly as interactive. By using the clipping modules, the user can interact with a small subset of the domain initially, and then, once the desired ranges have been determined, can pass the entire volume in to be segmented.

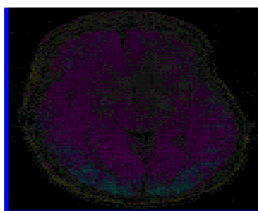
The next stage in the pipeline is the segmenting module, VectorSeg, which accepts the clipped volume fields as input. The interface for this module is a 2D array of range sliders for specifying grey-scale values of materials from each input volume. In this case, the module accepts two fields as input and segments for five distinct materials (in our case skin, bone,fluid, grey matter and white matter). If a voxel's vector is contained within the space spanned by the ranges of that material, that voxel is set to be "on" for that material.

The result of the vector segmentation module is a field of n-bit numbers, with one bit corresponding to each material. This field is passed on to the "BitVisualize" module, which creates a volume of colored points corresponding to each voxel's identified material. The volume of points is a "quick-and-dirty" method of volume-rendering the segmented data, and is a fast way to look at the segmentation results.

As in other applications, the Salmon module is used for rendering and interacting with the data.

Figure 14. Automatic segmentation of a portion of a brain using the vector segmentation dataflow module.



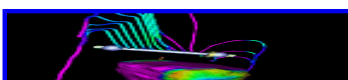


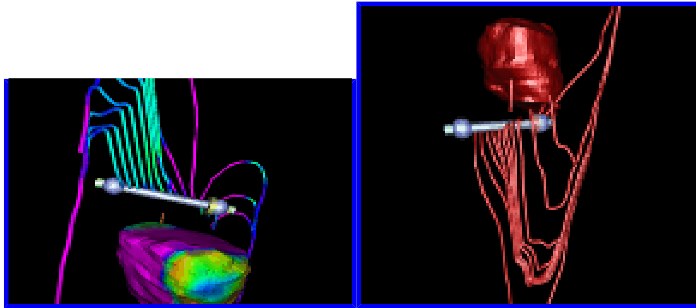
Cardiac defibrillator electrode design

Every year, 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 [13,14], for simulating both the endogenous fields of the heart and applied current sources (defibrillation devices). We are also able to simulate the multitude of electrode configurations, electrode sizes, and magnitudes of defibrillation shocks, via computer modelling. 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.

To meet this challenge, we are using SCIRun to design internal defibrillator devices and measure their effectiveness in an interactive graphical environment. Using SCIRun, engineers are able to design internal defibrillation devices, 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 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 design in terms of distribution of electrical current flow and density maps of current distribution.

Figure 15. Two visualizations of results from an internal defibrillation simulation showing the complex electrical current flow in a region near the heart. In the first figure, the colors indicate the magnitude of the vector current field, with purple representing highest magnitude and red representing the lowest magnitude.





Work in Progress

The SCIRun system is currently being expanded in many different directions and its modules enhanced. As SCIRun is applied to new problems in computational fluid dynamics, mesh visualization, automatic segmentation, vector field visualization and environmental science, it will be extended to meet the demands of each of these applications. Some of the planned extensions include:

- More control via steering
- Distributed computing
- Optimizations for larger problems
- Further Expansion of the Visual Programming Paradigm

A key aspect to the success of SCIRun is the advice and feedback from people who use it to solve real problems. We continually seek input from these "real users" and make changes and improvements to suit their needs. In addition, work has commenced in porting SCIRun to many new architectures, including Sun, IBM, Cray and HP.

Conclusions

We have outlined the architecture of the SCIRun scientific computing development system. This system employs a dataflow programming model and visual programming to simplify the tasks of creating, debugging, optimizing and controlling complex scientific simulations.

As efficiency issues are addressed, the dataflow programming model is being successfully applied to more problems and to larger datasets. This environment has been applied to real problems in computational medicine, and has proven to be a useful paradigm for investigating computational engineering problems.

Acknowledgements

This work was supported in part by awards from the Whitaker Foundation, the NIH, the NSF and the DOE. The authors would like to thank K. Coles, J. Purciful, 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.

DEMO

We now allow the reader to test the SCIRun computational steering system on a small application problem. The demo version works only on SGI computers with R4000 processors and higher, and with Irix 5.2 operating system or newer. Obviously faster processors and graphics hardware will allow for greater interactivity. For this version, multiple CPU's on an Onyx will provide a little more than single processor performance. To save a copy of the SGI executable on your disk, follow these instructions:

Instructions:

- To get the SGI executable off of the CD:
- When you are ready, click here and save the page to disk as a file called "demo.uu" (or whatever you prefer).
- "cd" to the directory where you downloaded the file.
- type: **uudecode demo.uu**
- type: **uncompress SCIRunDemo.Z**
- Make sure that the executable (SCIRunDemo) is exactly **2153632** bytes.
- You can now delete demo.uu.
- Make sure that you are sitting at an SGI and type ./SCIRunDemo and have fun! Click here for instructions on how to use it.

Alternatively, the demo program is available via FTP movie of portions of the demo is included. Due to space constraints, this movie is extremely small, short and somewhat fuzzy; see the live version for a more accurate reflection of the capabilities of SCIRun.

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. Upson, T. Faulhaber Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. "The Application Visualization System: A computational environment for scientific visualization." IEEE Computer Graphics and Applications, pages 30-42, Jul. 1989.
3. D.S. Dyer, "A dataflow toolkit for visualization," IEEE Computer Graphics and Applications, pages 60-69, July 1990.
4. C. Upson et al., "Future directions of visualization software environments," in SIGGRAPH '92 Panel Proceedings, 1991.
5. B. Lucas et al. "An architecture for a scientific visualization system," in Proceedings of Visualization '92, pages 107-114, Oct. 1992.
6. C. Williams, J. Raruse, and C. Hansen. "The state of the art of visual languages for visualization," in Proceedings of Visualization '92, pages 202-209.
7. D. Song and E. Golin. "Fine-grain visualization algorithms in dataflow environments." in Proceedings of IEEE Visualization '93, pages 126-133, Oct. 1993.
8. W.E. Lorensen and H.E. Cline, "A high resolution 3D surface construction algorithm," in SIGGRAPH '87 Conference Proceedings, pages 163-170, Jul. 1987.
9. D.B. Conner, S.S. Snibbe, K.P. Herndon, D.C. Robbins, R.C. Zeleznik, and A. van Dam. "Three-dimensional widgets." Computer Graphics (1992 Symposium on Interactive 3D Graphics), pages 183-188, Mar. 1992.
10. R.C. Zeleznik, K.P. Herndon, D.C. Robbins, N. Huang, T. Meyer, N. Parker, and J.F. Hughes. "An interactive 3D toolkit for constructing 3D widgets." Computer Graphics (SIGGRAPH '93 Proceedings), pages 81-84, Aug. 1993.
11. K.P. Herndon and T. Meyer. "3D widgets for exploratory scientific visualization." Proceedings of UIST '94 (SIGGRAPH), pages 69-70, Nov. 1994.
12. J.T. Purciful. "Three-dimensional widgets for scientific visualization and animation." masters thesis in preparation, Dept. of Computer Science, Univ. of Utah, 1995.
13. C.R. Johnson, R.S. MacLeod, and M.A. Matheson. Computer simulations reveal

complexity of electrical activity in the human thorax. Computers in Physics May/June, pp. 230-237, 1992.

14. C.R. Johnson, R.S. MacLeod, and M.A. Matheson. "Computational medicine: Bioelectric field problems." IEEE COMPUTER, pages 59-67, Oct. 1993.

Steve Parker - sparker@cs.utah.edu and Chris Johnson - crj@cs.utah.edu

---

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
SC '95, San Diego, CA
© ACM 1995 0-89791-985-8/97/0011 $3.50