BUILDING DOMAIN-SPECIFIC ENVIRONMENTS FOR COMPUTATIONAL SCIENCE: A CASE STUDY IN SEISMIC TOMOGRAPHY*

JANICE E. CUNY , ROBERT DUNN , STEVE T. HACKSTADT , CHRISTOPHER HARROP , HAROLD HERSEY , ALLEN D. MALONY , AND DOUGLAS R. TOOMEY

Abstract. We report on our experiences in building a computational environment for tomographic image analysis for marine seismologists studying the structure and evolution of mid-ocean ridge volcanism. The computational environment is determined by an evolving set of requirements for this problem domain and includes needs for high-performance parallel computing, large data analysis, model visualization, and computation interaction and control. Although these needs are not unique in scientific computing, the integration of techniques for seismic tomography with tools for parallel computing and data analysis into a computational environment was (and continues to be) an interesting, important learning experience for researchers in both disciplines. For the geologists, the use of the environment led to fundamental geologic discoveries on the East Pacific Rise, the improvement of parallel ray tracing algorithms, and a better regard for the use of computational steering in aiding model convergence. The computer scientists received valuable feedback on the use of programming, analysis, and visualization tools in the environment. In particular, the tools for parallel program data query (DAQV) and visualization programming (Viz) were demonstrated to be highly adaptable to the problem domain. We discuss the requirements and the components of the environment in detail. Both accomplishments and limitations of our work are presented.

1. Introduction. Programming environments and tools for parallel scientific computing have been developed, for the most part, in isolation from the requirements of the computational science domain in which they are to be used. The obvious reason is that tremendous efforts have been made in recent years to advance technology for parallel scientific problem solving in general. Developed technologies include parallel languages (e.g., HPF [9]), communications libraries (e.g., MPI [1]), parallel performance analyzers (e.g., Paradyn [7]), parallel debuggers (e.g., Ariadne [20]), and visualization systems (e.g., ParaGraph [15]). Solving a particular computational science problem, though, involves a combination of several technologies with a domain-specific purpose. That is, an "environment" for a computational science application must address problem solving requirements that are unique to that application domain. The process of creating such an environment requires collaboration between application scientists and computer scientists. The application scientists must establish requirements for problem investigation and evaluate the environment as its implementation proceeds. The computer scientists must fashion available technologies to address the requirements and possibly even develop new infrastructure to complete the environment.

Until recently, there has been little interest in understanding this collaborative process. As suggested above, this is in part due to the computer scientists' focus on advancing the state of general HPC technology. It is also in part due to the application scientists' focus on achieving high application performance. Performance is still an important concern, but the increased sophistication of scientific problem solving is demanding more robust, integrated support from computational tools of diverse

* Computational Science Institute. This work was supported by grants from ARPA (Air Force No. AF-3062-92-C-135, Army No. DABT63-94-C-0029), the Office of Naval Research (N00014-91-J-1216) and grants from the National Science Foundation (Academic Research Infrastructure (No. STI-9413532), Metacenter Regional Alliance (No. ASC-9523629), National Young Investigator (No. ASC-9457530), OSD and CISE/ASC (No. ASC-9522531), OCE (No. OCE-9106233), and Faculty Award for Women (No. CCR-9023256)). The research was done within the Computational Science Institute at the University of Oregon.

types. Until now, neither group has been willing to invest the time required to develop this support collaboratively. The challenge now for the application scientist is to develop environments that meet their domain-specific requirements for large scale data management, heterogeneous data analysis, model-based visualization, runtime program interaction, and coupled simulations. The challenge for the computer scientist is to intercede in specific projects while, at the same time, developing tools and infrastructure that will allow the easy customization of subsequent application-specific environments.

We believe that domain-specific environments for computational science represent a challenging research problem area for computer science. It includes many of the motivations and characteristics of problem-solving environments (PSE's) [10], but with an important difference: the leading-edge science applications we are concerned about are exploratory and experimental in nature. Most PSE's target applications which are "well understood and standardized" [10]. Our applications demand domain-specific support that can adapt to changing requirements. Thus, it is our belief that

> computer science research needs to develop methods, tools, and infrastructure that utilize capabilities of programmability, extensibility, and interoperability to build domain-specific environments for leading-edge scientific applications.

In this paper, we report on our experiences in building a computational environment for tomographic image analysis for marine seismologists studying the formation and structure of volcanic mid-ocean ridges. Before we began our collaboration, the seismologists had an ad hoc environment, running on a workstation, using sequential Fortran 77 and MatLab. This was so computationally limiting that only a fraction of data collected at the East Pacific Rise (EPR) in 1988 had been analyzed in 1996 when we began our work. The "ideal" computational environment for this application is determined by an evolving set of requirements for scientific analysis. It includes support for high-performance parallel computing, large data analysis, model visualization, and computation interaction and control. We discuss the seismic tomography application and its requirements in §2. This discussion typifies one aspect of the collaborative process where the application scientist explains the problem domain, describes the scientific models and methods used, and suggests possible avenues for computational exploration.

In §3, we put on our "computer science hat" and discuss how the seismic tomography requirements influence the consideration of infrastructure and tools used to build a computational environment. In particular, we describe two of our visualization tools – DAQV and Viz – that are designed to be flexible and extensible, making it easy for them to adapt to the domain-specific aspects of an application.

Our current computational environment for seismic tomography is presented in §4. For the geologists, the use of the environment has led to fundamental geologic discoveries on the East Pacific Rise, the improvement of parallel ray tracing algorithms, and a better regard for the use of computational steering in aiding model convergence. The computer scientists received valuable feedback on the use of programming, analysis, and visualization tools in the environment.

The work presented here is experiential. No general principles on building domain-specific environments for computational science were necessarily sought, nor fully discovered. However, we believe that reports of lessons learned and feedback on the collaborative process are valuable. We discuss these lessons and our plans for future development of the environment in §5. Our conclusions are presented in §6.

2. Seismic Tomography: Needs for a Computational Environment. Mid-ocean ridges are the primary sites for transferring heat and material from the Earth's interior to its surface. Much of the planet's generation of magma occurs at these ridges where, each year, 20 km^3 of new oceanic crust is formed as magma rises up from the mantle to form seafloor volcanoes. Magma chambers – volumes of molten rock that cap the magma plumbing system and convectively force the circulation of seawater through newly formed crust – play a central role in the exchange of heat and mass at these ridges. Our work aims at understanding the magmatic, hydrothermal, and tectonic processes involved. We use seismic tomography to construct 3D models of the seismic velocity at points in the crust and underlying mantle. Since seismic velocities are sensitive to elastic anisotropy, temperature, the presence of partial melt, and variations in porosity caused by cracking and the circulation of seawater, images of seismic velocity structure can be used to infer physical properties.

Seismic tomography is used to construct 3D images from seismic wave information recorded by sea-floor seismometers. The technique, similar to that used in medical CAT scans, uses recorded arrival times of acoustic energy from explosive or earthquake sources. Energy from the sources propagates downward into the Earth and is then refracted back to the surface by the positive velocity gradient of the crust and mantle. The travel time of a seismic wave from its source to a receiver depends on the velocities sampled by the wave along its path. The computation of velocity structure from travel-times is achieved by search of the parameter space, with each evaluation requiring first a forward propagation of ray paths and then an inverse calculation.

For the forward calculation, we determine synthetic travel times and seismic ray paths through a velocity model represented as a 3D grid of nodes [27]. The grid is used to construct a graph by imposing an adjacency template (or "forward star" of edges) at each of its nodes. The edges are assigned weights equal to the calculated travel times between their end points for the current velocity model. An adaptation of Dijkstra's shortest path algorithm [23] is used to find the minimum travel times between each seismic receiver location and all other nodes. Since calculated ray paths are restricted to passing through the discrete nodes of the velocity model, the accuracy of the method is limited by angle discretization and can be improved by increasing node and edge densities.

The inverse calculation perturbs the velocity structure to better match the observed travel-times. It solves a large, sparse system of linear equations relating velocity structure to travel-time data and a set of prior constraints. This system can be written as

$$G\Delta m = \Delta d$$

where m is a vector of model parameters representing perturbations to the initial velocity model, d is a vector which includes the differences between observed and calculated travel times, and G is a matrix of partial derivatives relating the two. The inversion solves the equation

$$\Delta m = G^{-1}\Delta d.$$

Since the accuracy of the forward calculation depends on the nodal density of the velocity model, the number of elements in m can exceed the number of travel time observations by as much as two orders of magnitude, resulting in an ill-conditioned inverse problem. We remedy this by adding equations to G to represent model parameter constraints that control the codependence of parameters. Thus we regularize the inverse problem to improve the stability of its solutions.

The basic computation as described, however, is only part of a much larger analysis process that can take several months. The full analysis requires extensive, domain-specific input from a geoscientist. In a sense, the scientist "steers" the generation of the final model, examining intermediate results to check the validity of the inputs, establish parameter values, optimize the calculation, test hypotheses, and determine the robustness of the final model. The goal of our environment is to support this total process.

To build appropriate support tools, we broke the overall problem solving process for seismic tomography into a series of seven steps and generated a list of domain specific requirements for each of those steps.

Step 1: Preprocessing of Data.
Initially the seismic data is captured as a set of records, one from each seismometer. Each record contains a stream of readings which is first separated and associated with individual source events. In addition, the precise location of the instruments on the seafloor is calibrated and all of the data is appropriately formatted in input files.

Requirement: Data Reformatting and Management Tools. Currently, preprocessing is done in an ad hoc manner with separate, hand coding of the reformatting. Ideally, the environment should provide more general tools for processing seismograms, reformatting the data, and managing the resulting input files. Since, however, the entire preprocessing step had already been completed for our current data sets, this aspect of the environment was given low priority. It is not currently implemented and will not be discussed further here.

Step 2: Input Initial Model.
The initial model is described by five input files giving the dimensions of the model and its initial slowness (inverse velocity) and anisotropy values.

Requirement: None. Initial models come from a collection of pre-existing models as determined by prior studies. Ours come from an earlier experiment that was able to produce fairly accurate 1D velocity profiles [29]. The environment could provide some facility for managing the associated input files for the initial model, but typically there are only a few such files and the needed support is not significant.

Step 3: Verification of Experimental Setup.
Here the "setup" phase of the computation is run and obvious errors are removed. First, we verify that the experimental geometry is consistent by checking spatial relationships between sources, receivers, initial slowness and perturbation models, bathymetry, and anisotropy. Then starting model is checked for consistency against the velocity-depth function.

Requirement: Visualization of 3D Spatial Relationships. We need the ability to display volumetric images of the model parameters and to examine the intersections between data and model features.

Step 4: Validity Check and Optimization of Ray Paths.
Once the experimental geometry is set up, the user performs a series of single-iteration runs of the program, examining the output of each run. First, the seismic ray paths are checked to see that they are roughly correct. Ray paths associated with outliers are examined and the seismograms are visually inspected to detect misidentified arrivals. Then a derivative weight sum is computed to reveal the well- and poorly-sampled

portions of the velocity model in order to determine whether or not there is enough data coverage to image the desired volume. (If not, additional seismic data may be necessary.) Finally, runtime performance of the ray-tracing calculation is improved by "carving" out sections of the model space not sampled by seismic rays. This optimization is performed separately for each receiver.

Requirement: Visualization of Individual Ray Paths. In order to check the ray path calculations, it is necessary to visualize individual ray paths superimposed on the volumetric visualizations of model parameters.

Requirement: Data Management and Visualization of Seismograms. To remove or correct outlying data, it is necessary to interact with visualizations of ray paths and seismograms. For example, a graphical interface could allow selection of a particular source-receiver group and subsequent display of the seismograms. This has not been implemented.

Requirement: Ability to Slice through 3D Visualizations of Derivative Weight Sums and Display Isosurfaces. To fully understand the sampling quality of the data and how it varies with depth, visualization of the derivative weight sums requires the display of isosurfaces and slices through volumetric visualizations.

Requirement: Ability to Interactively Carve Out Regions of a 3D Visualization. The user will need to graphically interact with visualizations of ray paths superimposed on isosurfaces of the "carved" regions of the model, to adjust the regions until they are as large as possible without touching the ray paths.

Step 5: Verification of G Matrix.
In this step, the scientist "eyeballs" the G matrix to see if it appears to be filled correctly, checking such things as Does the number of columns equal the number of model parameters? and Does the number of rows equal the number of constraint equations? In addition, individual row or column elements are plotted and checked against expected values. This step is particularly useful when new constraints are added to test data against hypothesized models.

Requirement: Visualization of G Matrix. The G matrix is quite large. In our current experiment, we have 40,039 parameters and 125,479 equations, represented in a sparse matrix having 757,343 nonzero elements (5,024,053,681 total elements). Obviously, the matrix cannot be examined directly. Instead the scientist views an image of the matrix in which values (or groups of contiguous values) have been replaced by pixels. Even that image is too large and the user must have the ability to pan and zoom across it, eventually moving to a level where it is possible to examine specific values.

Step 6: Establish Model Constraint Parameters.
This is the heart of the process where the scientist searches for the velocity model that best fits the data by performing a series of "experiments," each involving one or more iterations of the basic computation. Initially, single iterations are used to roughly establish the range of possible values for the model constraint parameters. The final runs are done with multiple iterations (up to 6 at a time). After each set, the scientist uses domain knowledge to readjust the constraint parameters, hopefully moving towards a better fit of the data. Eventually, he/she chooses a final model or class of models from the probable candidates that achieves the best fit to the data without undo complexity (Occam's razor).

Requirement: High-performance Computing. The geologists involved in this project originally approached the computer scientists because of performance issues. Their code was slow; a typical iteration took as much as six hours, making it difficult to explore enough parameter settings to produce a robust model. They were also limited in the size of the data sets that they could image. As the code was parallelized and computational times significantly improved, however, the inability to interact efficiently with the model and data spaces became more of a limitation.

Requirement: Computational Steering. Computational steering is the ability to interact directly with an executing program, modifying its data or control flow, in order to improve performance or accuracy. As the performance of the tomography code improved, much of the analysis that had been performed off-line could be supported on-line. This implies a mechanism for interacting with the distributed code and data in a manner which is meaningful to the seismologist.

Requirement: Data Management. In exploring models, the scientist examines data from multiple runs of the basic computation, each of which results in a candidate velocity model. To support this activity, the user should be able to browse through a history of these computations along with their inputs, parameters, and quantitative evaluations such as measure of data misfit. This aspect of the environment has not yet been implemented.

Requirement: Visualization. In examining the models, the scientist simultaneously considers multiple types of data and their relationships. All of the visualizations discussed above are needed, interoperating together with a common interface.

Step 7: Confidence Testing.
After selecting a model that best fits the data, the scientist evaluates its robustness. This is done by again examining the derivative weight sums to determine coverage; that is, to determine which parts of the models are well-sampled. In addition, the scientists engages in "hypothesis testing" to see if imaged features are reliable over a range of parameter values. One method of hypothesis testing is to remove suspect features from the final model and then reinvert the data using the altered model as a starting model. Removed features that are not supported by the data will not be reimaged.

Requirement: Data Management, Visualization, and the Ability to Graphically Interact With Visualization. As above.

In summary, then, the requirements of the seismic tomography environment can be classified as improved performance and sophisticated, interactive visualization and management of large amounts of data.

3. Technology for Domain-Specific Environments. There is now suitable HPC hardware and software technology for developing computational science applications, but scientists often lack sufficient background in parallel processing to understand and apply that technology. Experience has shown that it is difficult to make programming and analysis tools easy to use [5]. Generally, difficulties arise when the scientist's conceptual and operational models of the application are not well supported by the tool. A simple example was seen in early versions of ParaGraph [15] where displays of communication behavior clearly showed performance bottlenecks, but users found it difficult to relate those bottlenecks back to specific source code. Support for this was later added in tools derived from ParaGraph, as were other capabilities for tailoring output more specifically to application needs.

In our work on parallel performance analysis and debugging, we have addressed the problem of making tools more specific to applications in three ways.[1] First, we designed tools that support programming and computational abstractions that can be easily targeted to an application. As an example, the event-based Ariadne debugger matches user-specified models of intended program behavior against actual execution behavior as captured in event traces [20]. Ariadne is, in essence, "programmed" to work with an application by specifying appropriate behavior models and enabling instrumentation. Second, since complete functional requirements are difficult to anticipate, we constructed tools that could be extended with new functionality. This is principally an issue of tool architecture. For the $\mathcal{T}$ (TAU, Tuning and Analysis Utilities) program analysis environment [4], as an example, we developed a toolkit architecture where individually distinct tools work together based on a common system infrastructure. The benefit of such an architecture is that new tools can be added easily as long as they are consistent with the infrastructure; portability is also facilitated in this way. Third, we developed tools to interoperate by specifying their programming and communications interfaces. Again, $\mathcal{T}$ provides a good example of this feature as tools can evoke integrated behavior via synchronized, communicated actions [4]. Often the need for interoperability also necessitates extension of the architecture infrastructure, as in the case of $\mathcal{T}$'s program and interaction control infrastructure [26].

The three general features of programmability, extensibility, and interoperability have proven to be effective in our performance and debugging tools [22]. We speculate that these same capabilities will be useful in addressing the domain-specific aspects of computational science applications. Recommendations from the PSE community – arguing for new environment architectures, supporting software infrastructure, and tool components – are consistent with this view [10]. For our environments, they will be even more critical because the evolving nature of more exploratory work requires tools that can not only accommodate domain-specific abstractions but also adapt to changing requirements.

We have developed two tools – DAQV and Viz – that include design features enabling their better use in domain-specific computational environments. These tools are representative of a class of tools whose design and development, we believe, will become more interesting for research study. Indeed, other papers in this workshop report tools or infrastructure with similar concerns. We describe the DAQV and Viz tools below and then discuss their use in the seismic tomography environment in §4.

3.1. DAQV: Distributed Array Query and Visualization. During the execution of a parallel program it is often necessary to inspect and modify the values of parallel data structures that are decomposed across the processing nodes of a parallel machine.[2] In general, users do not know anything about data decompositions, symbol tables, or the number of processors involved. How can we provide access to distributed data structures at a level which is meaningful to them? *DAQV* addresses this problem, providing high-level access to parallel, distributed arrays for the purpose of visualization and analysis by external tools. DAQV has three primary design objectives:

[1] Other researchers have followed similar approaches.
[2] In particular, this is necessary for computational steering.

- A logical, global view of parallel, distributed data;
- A simple, many-to-one client/server model for tool interaction; and
- High-level language portability and compiler-independence.

In languages such as High Performance Fortran (HPF), the programmer views distributed arrays as part of a global name space, accessible without concern for which processor stores the data. We developed DAQV for HPF to expose this high-level, global view of data outside the program. DAQV essentially turns an HPF program into a distributed data server for external client tools. It does this through a user-extendable library of HPF routines for data access, utilizing the HPF compiler to produce correct code for distributed data retrieval; DAQV portability and retargetability is thus provided by the compiler. A socket-based server interface and protocol is implemented for external communications.

DAQV supports two different operational models: *push* and *pull*. The push model constitutes the simplest and least intrusive way to access distributed arrays from an external tool, called a *data client*. The push model is invoked by DAQV subroutine calls placed into the HPF source code, allowing the programmer to (1) indicate the DAQV model (i.e., push or pull) to be used in the program, (2) register distributed arrays with DAQV, (3) set parameters for communicating with a data client, (4) make DAQV connect with a data client, and (5) send the data values of a distributed array to the data client. In contrast, the pull model allows an external tool, called a *control client*, to determine when and what distributed array should be accessed. DAQV allows control client to direct program execution and to configure and initiate array transfers to data clients.

The DAQV tool includes all three of our desired features: its data access capabilities can be programmed; its data access library and client toolset can be extended; and its well-defined client/server protocol based on standard interprocess communication supports interoperability. For a more detailed discussion of DAQV, see [12].

3.2. Viz: A Visualization Programming System. Creating meaningful visualizations for use in scientific problem solving is difficult [28]. General purpose visualization tools, such as Iris Explorer [24], AVS [21], and Data Explorer [6], have been well received, but do not always provide the flexibility and robustness required for visualization design and specialization. *Viz* was created to support rapid visualization prototyping in a system that could be extended with application-specific visualization abstractions. Our original application domain was parallel program and performance data visualization [11, 13, 16, 17].

Viz is a visualization programming environment built on an interpreted, high-level language (Scheme [19, 8]) with object extensions (Meroon [25]). It embeds a 3D graphics library (Open Inventor [30]), adding higher-level abstractions for glyphs, composition, and data representations. Viz supports a visualization process model called *data reactivity* that uses mechanisms that have proved to be important in visualization design and operation (e.g., constraints, controlled data flow, dynamic analysis, animation, and interaction).

A developer using Viz can program domain-specific analysis and visualization abstractions through procedural abstraction and object inheritance. Viz also provides mechanisms for "wrapping" visualizations into stand-alone tools that can communicate with other components (e.g., data sources), enabling interoperability in a larger environment. In addition to extension through new abstractions, Viz system comes with powerful facilities for extending functionality through foreign function interfaces [14]; the Open Inventor API, the socket support, and a Viz module for Iris Explorer

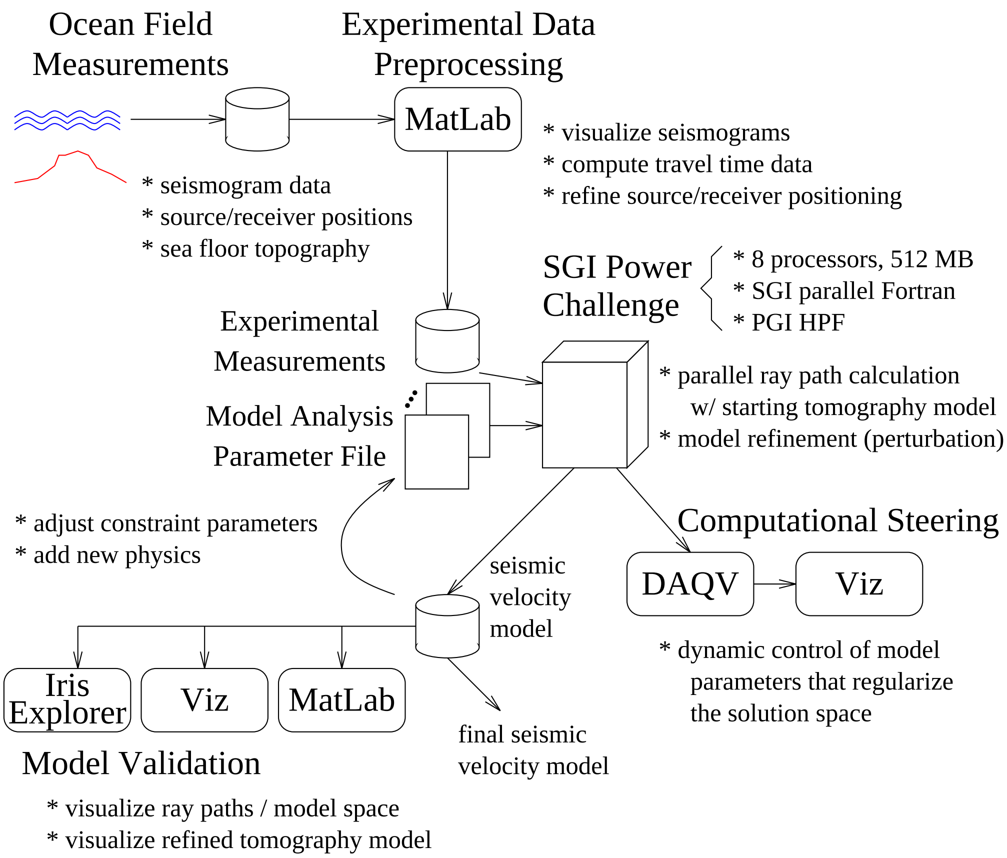


FIG. 1. *Seismic Tomography Computational Environment*

were implemented in this manner. We are actively defining new abstraction layers to support, for instance, interaction, animation abstractions, and distributed operation. For a more detailed discussion of Viz, see [18].

4. A Domain-Specific Environment for Seismic Tomography. The current experimental environment for the marine seismology application is shown in Figure 1. This environment has evolved from a workstation platform running sequential Fortran 77 (F77), using MatLab [2] for data analysis and visualization, to a high-performance multiprocessor platform. Our platform, an SGI Power Challenge (SGI-PC) with 8 processors, provides parallelizing compilers and parallel languages, with support for dynamic program interaction and 3D visualization.

Our first objective was to increase the speed of the tomography code. The forward calculation (as described in §2) was so computationally intense that it prevented the seismologists from performing short-term analyses of their data. We trivially parallelized it, letting each processor compute the ray paths for a subset of the receivers. A near-linear speedup was achieved.[3] For the inverse calculation, we relied on the

[3]Since we have available three SGI-PC machines, totalling twenty-two processors, we experimented with a cluster implementation of the "receiver-parallel" method using PVM. Unfortunately, the cluster interconnections at the time were only 10 Mb/s Ethernet links and the data exchange overhead needed for the inverse calculation defeated the extra parallelism benefit.

SGI F77 compiler to automatically parallelize matrix operations. The improvement in speed was significant, reducing the time for a single, forward and inverse iteration from approximately 6 hours to 20-30 minutes.[4] These improvements allowed more rapid analysis and the consideration of considerably larger data sets.

The increased performance shifted the computational analysis bottleneck from model generation to model evaluation. Because the seismologists must evaluate the models produced by each set of iterations in order to readjust the constraint parameters for the next run, we began to consider support for this part of the analysis. Two approaches were pursued: enhanced visualization and on-line analysis. Originally, models were evaluated using 2D visualizations produced by MatLab. The seismologists could gain a more comprehensive view of the models and more easily adjust constraints with 3D volumetric visualizations. We applied NAG's Iris Explorer [24] to generate model isosurfaces, but found that data model and formatting incompatibilities between Iris Explorer and MatLab made it difficult to merge the two tools.[5] We began using Viz to overcome these difficulties, and quickly found that its programming flexibility and abstraction capabilities allowed the seismologists to be even more creative in their visualization ideas. The visualizations available in the environment are shown in Tables 1, 2, and 3.

To support model analysis on-line, we needed a means of interacting with the running computation. DAQV provided this means, but required a HPF program. We reimplemented the tomography code in HPF and used the *pghpf* HPF compiler from Portland Group, Inc. for compilation on the SGI-PC machine. The straightforward receiver-parallel approach was not especially effective in HPF's data-parallel programming model, and much of our earlier performance benefit was lost. However, anticipating future work on scalable, data-parallel algorithms (see §5), we proceeded in order to gain experience using DAQV. DAQV made it easy for the seismologists to access various model data at runtime and pass it to Viz-based visualizations. Because DAQV also supports program data alteration, steering operations to control model constraints are also possible. Using DAQV in this manner, the graphical user interface of the control client allows the seismologist to specify input data, constraint parameters, and the run-time options needed to launch, start, stop, and continue the execution of the program. The Viz-based data clients let the seismologist turn on and off specific visualization features to observe the model space, slowness fields, and travel-time fields. Thus, for example, the thresholds of the slowness and travel-time isosurfaces can be changed via a slider button to effect steering control.

Using the Environment: A Scenario

We describe a typical scenario in using the seismic tomography environment in developing an image, relating the discussion back to the seven steps listed in §2. We assume that the raw data has already been preprocessed as in Step 1.

To start an analysis session, the scientist launches the environment and is presented with the graphical user interface (GUI) of the control client. This GUI is used in specifying the input data files and the initial values for model parameters as in Step 2. The input data files contain all of the preprocessed information on sources, receivers, observed travel-times, bathymetry, starting velocity model, perturbation model grid size, and forward star templates. The initial values for the model parame-

[4] We continue to work on performance improvements.

[5] Iris Explorer also lacked some basic graphics production requirements for seismic image analysis (filled contours, labels, publishable quality printing).

ters include such things as the smoothing and damping constraints. At this time, the user also specifies the host(s) on which the program is to be executed and the total number of processors to use. He/she then clicks on "Go" to launch the tomography analysis application.

As the application starts up, textual output concerning the acceptance of the data is scrolled in a large text area at the bottom of the control client panel. If errors are indicated, the scientist exits the environment to correct the problem off-line. Once the input data has been successfully loaded, the utility of the text window diminishes, and the user relies upon the visualizations provided by the data client to guide the remainder of the analysis.

After loading the input data, the program continues until it reaches the first DAQV yield point, where the seismologist examines the experimental geometry as in Step 3. This is done with 3D visualizations as shown in Table 1 A,D. The sources and receivers are represented by different colors and geometric shapes (spheres and cones, respectively) for easy identification. The user can interactively rotate the pictures to view spatial relationships from any desired perspective. When the geometry is correct, the seismologist uses isosurface visualizations of the starting velocity model as shown in Table 1 D to verify that the starting model is consistent with the 2D plot of the velocity-depth function and cross-sections produced off-line using MatLab (Table 1 B,C). Any errors detected are again corrected off-line. To proceed, the user clicks on the "Continue" button on the control client panel.

At this point, the seismologist is on Step 4, checking the validity of the ray paths, removing or correcting data outliers, and optimizing their calculation. The program encounters the next DAQV yield point when it enters the ray-tracing code. The seismologist can then step through the calculation, observing the propagation of paths and their corresponding travel-time wave front as shown in Table 2 A,B. These visualizations can be used to verify that the ray paths, for example, sample the correct regions of the model, do not intersect "carved" regions, and do not propagate along obviously non-physical trajectories (ray paths should propagate outward as a roughly spherical "ball of hair"). When the user is satisfied that the ray-tracing appears to be working properly, he/she again presses "Continue" and the program proceeds uninterrupted until the ray-tracing has finished and the starting model misfit and a derivative weight sum have been calculated.

The user observes textual output from these short calculations and a visualization of the derivative weight sum as shown in Table 2 C. The user studies the isosurface visualization of the derivative weight sum to evaluate the sampling quality of the travel-time data, noticing regions which are under- or over-sampled. If the sampling is unsatisfactory, the calculation must be stopped and the problem resolved off-line. If it is satisfactory, the user inspects the "carved" regions of the model to make sure that they are not sampled. This is accomplished by overlaying the individual ray paths for each source and carved regions as in Table 2 D. The scientist can interactively alter "carved" regions of the velocity model by performing on-line mutations to program data.

In the next step, Step 5, the scientist advances the computation to the beginning of the inversion phase and views the G matrix. This is done with a 2D MatLab plot as in Table 3 A. This plot would reveal any obvious errors in loading the matrix with the medium derivatives and model constraint parameters.

Now the scientist is ready to begin the central process, the search through the parameter space described in Step 6. Advancing the computation to the end of the inversion, a model perturbation is produced and added to the initial velocity model. At this point, the seismologist uses another 3D isosurface visualization to view the new initial model and the perturbations produced by the inversion as shown in Table 3 B,C. Alternatively, the user may choose to view the resulting perturbation off-line using MatLab as shown in Table 3 D. Before proceeding with the next iteration, the user may modify the initial and/or the perturbational velocity model(s) interactively by selecting regions of the model space via the mouse. For example, the user may choose to leave the initial model unperturbed by zeroing the perturbations (a useful capability during hypothesis testing).

Each successive iteration of the ray-tracing and inversion phases of the tomography code proceed much like the first. The seismologist iterates the steps while searching the solution space for a physically plausible velocity model having the smallest data misfit. Once a model has been selected, the user moves into the final step of confidence testing, Step 7. He/she continues to use the environment as for Step 6.

5. Lessons Learned and Future Enhancements. Our tomography environment addresses many of the needs of the seismologists, significantly enhancing their ability to image the Earth's crust and mantle. Recently, for example, using an initial versions of the environment, they were able to successfully image a previously unknown section of the magma plumbing system beneath the EPR. Moreover, this feature was imaged using data that had existed since 1988 but could not be fully analyzed because of the computational requirements. Using our faster, parallel code enabled them to generate and analyze models much more rapidly, allowing them to "see, for the first time, trends in the data." These observed trends led them to suspect that an additional factor, seismic aniostropy, was significant. After incorporating anisotropy into their forward and inverse calculations, they were able to produce a robust, definitive image of a previously uncharted magma body. Thus, *the environment made a qualitative difference in the science that could be accomplished.*

In addition to this accomplishment, we learned a number of lessons in developing the tomographic environment:

1. **Programability, extensibility, and interoperability are important in the development of the domain-specific aspects of an environment.** Because of the exploratory manner in which the seismic environment was being developed and applied, it needed to evolve its functionality as new problem-solving ideas demanded new capabilities. These three design features proved valuable in allowing the environment tools to adapt. For instance, the ability to program data access in DAQV and visualizations in Viz gave the seismologists the convenience of changing focus for data analysis; often ideas for visualizing a different set of data could be tested in a few hours. Regarding extensibility, it is interesting to note that dynamic program interaction was not available initially; DAQV was extended with the ability to alter HPF program variables in response to the seismologists subsequent interest in steering the tomography computation once its performance increased. The support for interoperability in the tools made a "plug-and-play" style of environment construction, helping to reconfigure the environment when necessary.

2. **Dynamic program interaction is critical.** As the program's performance on a single iteration increased, it became possible to look at the imaging process, not as a series of isolated computer runs, but as a series of steering operations on a single (long) computation. Thus, we found it increasingly

important to allow flexible interactions with the running program in order to support what had previously been off-line analysis steps.

3. **Sophisticated visualizations were more important than the scientists originally gave them credit for.** Initially, the seismologists were interested only in speed improvements; they were content with 2D, off-line MatLab visualizations. As we produced more sophisticated, interactive 3D renderings, however, they became convinced of their utility. Currently, we are collaborating in the design of a variety of visualizations that will allow them to simultaneously view related data from a variety of perspectives.

4. **Speed critical for usage.** The seismologists were unwilling to use any program features that significantly impaired performance, even for testing purposes. This meant that even initial prototypes had to be carefully implemented if we were to get feedback.

The requirements of our environment are driven by the scientific discovery process. As we provide increasing support for the activities of the seismologist, the way in which they use the environment – and thus the requirements for the environment – change. Given the experimental nature of our application domain, this evolution will continue. Currently, we have a number of future upgrades in the design stages:

1. **Improved Visualizations.** As mentioned above, we are increasingly providing support for the model analysis activities of the seismologist.
   (a) *Color rays according to seismic phase.* Seismic phase roughly corresponds to the section of the Earth sampled by a ray along its path. Coloring individual ray paths according to their seismic phase will allow for easy identification and comparison, improving our ability to evaluate coverage.
   (b) *Contoured slices of 3D volumes.* Our current isosurface visualizations allow seismologists to view their volumetric data in three dimensions where previously, they had to infer 3D structure through 2D slices (cross-sections). The 3D isosurfaces improve the efficiency of their analysis but, since they do not show spatial changes in the volumetric data, they cannot become a replacement for the 2D contoured slices. We will thus provide a combination of slices and isosurfaces, allowing our users to visualize specified contoured slices (cross-sections) of their 3D volumes.
   (c) *On-line visualization of velocity depth function.* The scientists' ability to verify the correctness of the starting velocity model on-line will be enhanced with contoured vertical cross-sections of the model, providing similar information to that currently generated off-line by MatLab.
   (d) *Grid lines for 3D volumes.* The axes of the volumes visualized by our environment are currently labeled with user-adjustable tic marks. However, we have noticed that it is difficult for users to determine which tic marks line up with features inside the 3D volume. Thus we will allow tic marks to be extended to form grid lines throughout the volume.
   (e) *Multiple thresholds for isosurfaces.* The specification of multiple thresholds for isosurfaces will make it easier to visualize spatial changes in 3D volumes. Users will be able to specify any number of isosurface thresholds and view the resulting surfaces simultaneously.

(f) *Vector field visualizations.* A 3D vector field visualization will be implemented for depicting anisotropy data.

2. **Data Management.** The exploration of the parameter space to find the best fit model generates a considerable amount of intermediate data. Although much of it is discarded after the final model has been rendered, it is extremely valuable during the search process. A historical record of those results and how they were derived would be useful in guiding the analysis. To address this, we plan to create a graphical front-end to a database which will efficiently store and intelligently retrieve the intermediate data. Each model will be entered along with its inputs, associated misfit, and any users' comments.

3. **Support for Cross-Model Correlations.** Ultimately, scientists will need to correlate and validate data between a variety of theoretical and experimental models, placing an increasing emphasis on data interpretation, abstraction, and feedback. We hope to provide a data modeling language for issuing high-level statistical and behavioral queries on raw data for cross correlations between models, ultimately answering questions such as Does the seismic velocity field correlate with the predictions of a numerical model of heat flow? Is the density implied by the seismic velocity field consistent with observations of gravity? and What relationships exist between topographic features and density, as constrained by seismic and gravity data?

4. **Improved Flexibility/Interaction in Execution.** Currently, the analysis framework we provide is rigid in that it does not allow the user to dynamically choose the input data (sources and receivers) or to perform calculations (ray-tracing, derivative weight sum, inversion) on demand. For example, if a user wants to analyze only the data corresponding to a particular subset of receivers and/or sources, new input files must be created. This obviously restricts the flexibility of the environment and makes it difficult to perform "unusual" experiments. We will allow users to drive the analysis process through interaction with the visualizations. For example, in Step 4 of §2, users will be able to select a receiver and sources via the mouse and then click a button to invoke the ray-tracing calculation. Similarly, they will be able to select a receiver and invoke the computation of the derivative weight sum. Thus, the user will be able to conduct custom experiments without resorting to the modification of code or input data files.

5. **Improved Performance.** We continue to work on improving performance both for the parallel ray-tracing algorithm and the parallel inversion method. We want a parallel SVD (Singular Value Decomposition) method for computing pseudo-inverses of the G matrix, also yielding information and resolution matrices that will provide statistical information regarding the resolution and quality of the data. We are considering the use of SVD Pack [3]. We are also considering parallelization across our array of SGI Power Challenges (currently we run on a single cabinet).

6. **Portability**. We would like the environment to be portable for several reasons. First, it is important for the seismologists to share the environment with colleagues, who may have different computational platforms. Building tools based on standard libraries and interprocess communication facilities

enhances environment portability and retargetability. We would like both to "shrink-wrap" the environment to give to other scientists with seismic data sets, and to to provide an open environment than can be used potentially as part of a larger, heterogeneous system. Second, the computation itself must run on sequential and parallel systems. A Fortran-based, shared-memory program model is probably the best choice for this purpose, especially since small-scale shared-memory multiprocessor systems are becoming more prevalent. We chose to write a version of the computation in HPF, in part, to address scalability issues as the tomography models increase in size, while at the same time maintaining portability across different platforms.

6. Conclusions. In the past, computation's dominance in time and cost requirements has allowed tool designers to adopt a parallel system centered view, focusing their tools on the analysis of the parallel computation (e.g., for performance and debugging). The scientist's environment, however, has always been more heterogeneous, combining data gathering (static and dynamic), data analysis (manual and computational), data management (large and complex, real time and archival), and data visualization (still and animated, composed and real-time). As parallel computing technology changes (e.g., faster processors), so will the nature of the problem solving bottlenecks (e.g., from computationally intensive to data management or visualization intensive). Tool technology must be able to follow the changing focus and target the prevailing needs.

Tools that support domain-specific activities are an equally important part of the scientific computing environment as parallel programming, debugging, and performance tools. Moreover, we believe that as a research target, designing and developing domain-specific tools and environments for exploratory computational science offers a number of challenges: How can tools be customized and extended to meet changing application requirements? How can they work together to enable environment construction and management? and How can they integrate and incorporate the rich parallel computing infrastructure that has been developed?

We have taken one small step in trying to better understand the issues and aspects of building domain-specific computational environments. Our seismic tomography environment has successfully incorporated high-performance parallel computing with tools that can adapt to evolving problem-solving needs of the seismologists. The environment has already led to new scientific discoveries, and we believe this environment will continue to aid geophysicists in extracting the full extent of the science embodied by the seismic data sets. We look forward to continued work in domain-specific tool and environment research.

Acknowledgement. We would like to thank Amy Karlson for her work with Viz on labeling volumetric visualizations.

REFERENCES

[1] *Message Passing Interface (MPI)*. http://www.arc.unm.edu/workshop/mpi/mpi.html.
[2] *MATLAB: High Performance Numeric Computation and Visualization Spftware Reference Guide*, 1992.
[3] M. Berry, *Large Scale Singular Value Computations*, International Journal of Supercomputer Applications, 6 (1992), pp. 13–49.

[4] D. Brown, S. Hackstadt, A. Malony, and B. Mohr, *Program Analysis Environments for Parallel Language Systems: The TAU Environment*, in Proc. of the Workshop on Environments and Tools for Parallel Scientific Computing, May 1994, pp. 162–171.

[5] C. Cook and C. Pancake, *What Users Need in Parallel Tool Support: Survey Results and Analysis*, in Proc. of the Scalable High Performance Computing Conference, I. C. S. Press, ed., 1994, pp. 40–47.

[6] I. B. M. Corporation, *IBM Visualization Data Explorer, User's Guide, 2nd Ed*, Aug. 1992.

[7] B. M. *et. al.*, *The Paradyn Parallel Performance Measurement Tool*, IEEE Computer, 28 (1995), pp. 37–46.

[8] M. Feeley, *Gambit-C, A Portable Scheme Implementation*, Apr. 1996. Version 2.3.

[9] H. P. F. Forum, *High Performance Fortran Language Specification, Version 1.0*, Tech. Report CRPC-TR92225, Center for Research on Parallel Computation, May 1993.

[10] E. Gallopoulos, E. Houstis, and J. Rice, *Problem-Solving Environments for Computational Science*, IEEE Computational Science and Engineering, Summer (1994), pp. 11–22.

[11] S. Hackstadt and A. Malony, *Next-Generation Parallel Performance Visualization: A Prototyping Environment for Visualization Development*, in Proc. Parallel Architectures and Languages Europe (PARLE) Conference, July 1994, pp. 192–201.

[12] ———, *Distributed Array Query and Visualization for High Performance Fortran*, in Proc. of Euro-Par '96, Aug. 1996.

[13] S. Hackstadt, A. Malony, and B. Mohr, *Scalable Performance Visualization for Data-Parallel Programs*, in Proc. of the Scalable High Performance Computing Conference, I. C. S. Press, ed., May 1994, pp. 342–349.

[14] L. Hansen, *FFIGEN: Foreign Function Interface GENerator, release 1*, 1996. available from http://www.cs.uoregon.edu/~lth/ffigen.

[15] M. Heath and J. Etheridge, *Visualizing the Performance of Parallel Programs*, IEEE SOFTWARE, (1991), pp. 29–39.

[16] M. Heath, A. Malony, and D. Rover, *Parallel Performance Visualization: From Practice to Theory*, IEEE Parallel and Distributed Technology, 3 (1995), pp. 44–60.

[17] ———, *The Visual Display of Parallel Performance Data*, IEEE Computer, 28 (1995), pp. 21–28.

[18] H. Hersey, S. Hackstadt, L. Hansen, and A. Malony, *Viz: A Visualization Programming System*, Tech. Report CIS-TR-96-05, University of Oregon, Department of Computer and Information Science, Apr. 1996.

[19] IEEE, *IEEE Standard 1178-1990. Standard for the Scheme Programming Language*, 1991.

[20] J. Kundu and J. Cuny, *The Integration of Event- and State-Based Debugging in Ariadne*, in Proceedings of the 1995 International Conference on Parallel Processing, 1995, pp. II 130–134.

[21] B. Lucas, G. Abram, N. Collins, D. Epstein, D. Gresh, and K. McAuliffe, *An Architecture for a Scientific Visualization System*, in Proc. of Visualization '92, Oct. 1992, pp. 107–114.

[22] A. Malony, B. Mohr, P. Beckman, and D. Gannon, *Program Analysis and Tuning Tools for a Parallel Object-Oriented Language: An Experiment with the TAU System*, in Proc. of the Workshop on Parallel Scientific Computing, Oct. 1994.

[23] T. Moser, *Shortest Path Calculation of Seismic Rays*, Geophysics, 56 (1991), pp. 59–67.

[24] L. Numerical Algorithms Group, *IRIS Explorer User's Guide, Release 3.0*, 1995.

[25] C. Queinnec, *Meroon V3: A Small, Efficient, and Enhanced Object System*, Tech. Report URA 1439, École Polytechnique & INRIA-Rocquencourt, 1996.

[26] S. Shende, J. Cuny, L. Hansen, J. Kundu, S. McLaughry, and O. Wolf, *Event- and State-Based Debugging in TAU: A Prototype*, in Proc. of SPDT'96: Sigmetrics Symp. on Parallel and Distributed Tools, May 1996, pp. 21–30.

[27] D. Toomey, S. Solomon, and G. M. Purdy, *Tomographic Imaging of the East Pacific Rise Shallow Crustal Structure at* $9°30'n$, Journal of Geophysical Research, 99 (1994), pp. 135–24,152.

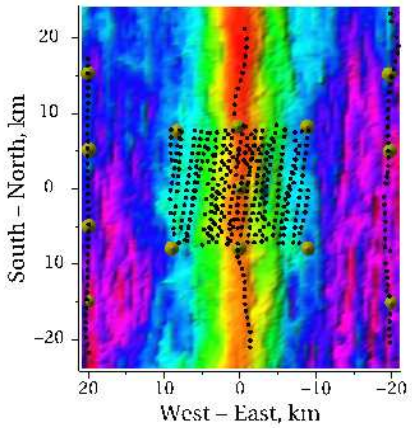
[28] L. Treinish, D. Butler, H. Senay, G. Grinstein, and S. Bryson, *Grand Challenge Problems in Visualization Software*, in Proc. of Visualization '92, I. C. S. Press, ed., Nov. 1992, pp. 366–371.

[29] E. Vera, *The Structure of* $0-$ *to* $0.2m.y.-$ *Old Oceanic Crust at* $9°n$ *on the East Pacific Rise from Expqanded Spread Profiles*, Journal of Geophysical Research, 95 (1990), pp. 15,529–15,556.

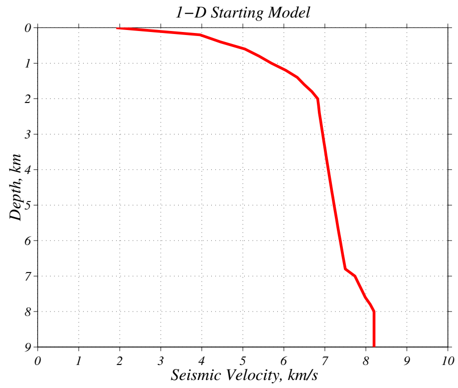
[30] J. Wernecke, *The Inventor Mentor*, Addison-Wesley, Reading, MA, 1994.

TABLE 1
Visualizations for Experimental Input Verification

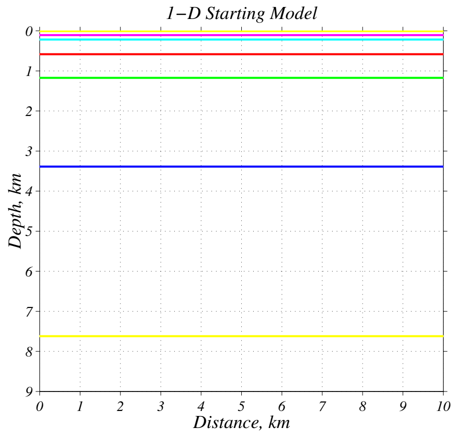
<table>
<tr>
<td>
A. Experimental geometry.



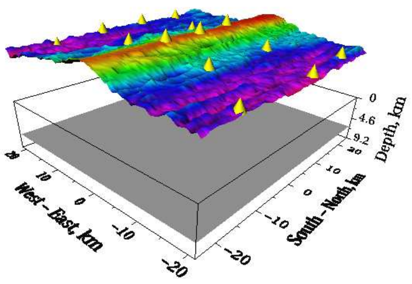
Allows users to verify the geometry of the sources, receivers, bathymetry, and model.
</td>
<td>
B. 2D MatLab plot of velocity vs depth for the 1D starting velocity model.



Used to verify the 1D starting velocity model prior to executing the tomography code.
</td>
</tr>
<tr>
<td>
C. Cross-section contour plot of the 1D starting velocity model using MatLab.

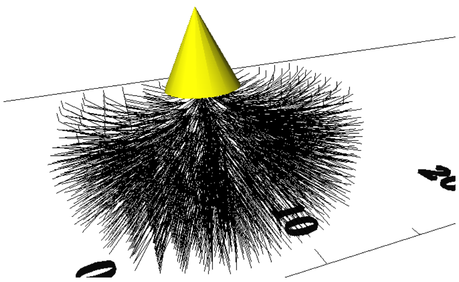
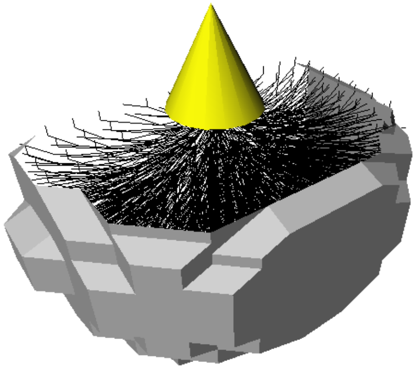
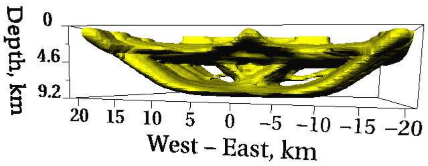


Enables users to see the layering of the starting velocity model prior to executing the tomography code.
</td>
<td>
D. Experimental geometry with a 3D isosurface visualization of the 1D starting velocity model.



Enables users to see the layering of the starting velocity model. A future enhancment will allow users to see the layering more easily by displaying vertical slices. Currently, vertical slices are only supported off-line using MatLab.
</td>
</tr>
</table>

Table 2
Visualizations of Ray Path Calculations

<table>
<tr>
<td>A. Ray path segments emanating from a receiver.

Users may roughly verify that ray path segments are being computed correctly.</td>
<td>B. Travel time wavefront emanating from a receiver.

Allows users to verify that the computed ray paths and corresponding travel time field are consistent.</td>
</tr>
<tr>
<td>C. Derivative weight sum isosurface.


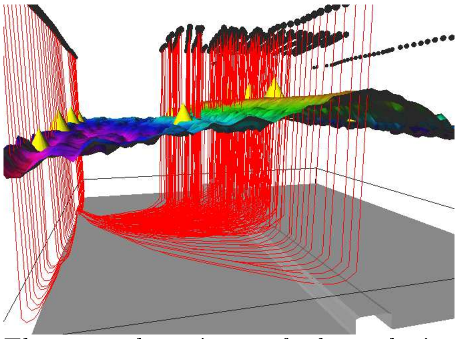
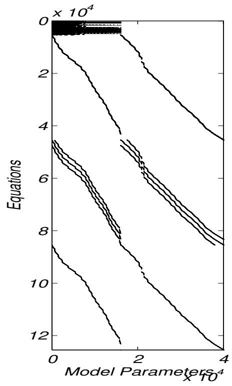
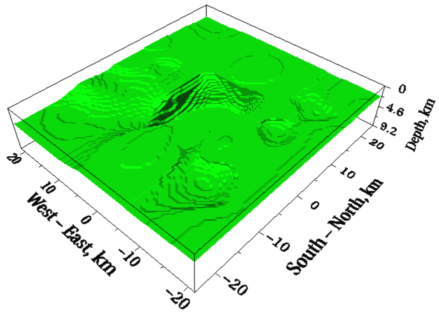
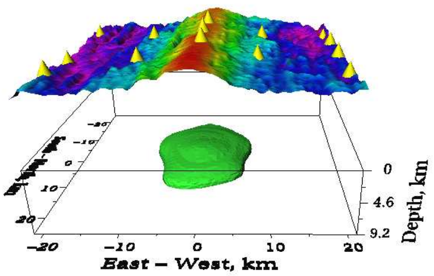
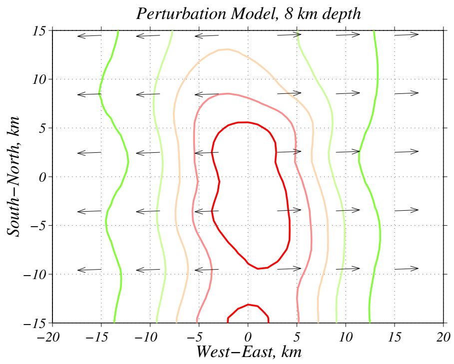
Reveals undersampled regions of the forward model space. Can be used to determine if there is enough data coverage to perform the intended experiment.</td>
<td>D. Individual ray paths and carving isosurface.

The carved regions of the velocity model may be compared to the individual ray paths to help verify that only unsampled regions are carved out of the velocity model.</td>
</tr>
</table>

TABLE 3
Visualizations for Model Refinement

<table>
<tr>
<td>A. 2D density plot of the sparse inversion matrix.

Used to detect errors in loading the inversion matrix. This visualization is only supported off-line using MatLab.</td>
<td>B. Initial model after perturbations have been added.

This may be compared to the perturbations to check for consistency.</td>
</tr>
<tr>
<td>C. Total perturbation to the starting velocity model.

Allows seismologists to identify seismic anomalies which may correspond to magmatic or other geologic features beneath the rise.</td>
<td>D. 2D horizontal slice of the velocity model perturbation and anisotropy.

MatLab plot depicting horizontal variations of the velocity model perturbation and anisotropy for a given depth (currently only supported off-line).</td>
</tr>
</table>