Published in final edited form as:
Neuroinformatics. 2010 March ; 8(1): 5–17. doi:10.1007/s12021-009-9061-2.

The Java Image Science Toolkit (JIST) for Rapid Prototyping and Publishing of Neuroimaging Software

Blake C. Lucas[1,2], **John A. Bogovic**[1], **Aaron Carass**[1], **Pierre-Louis Bazin**[3], **Jerry L. Prince**[1,3,4], **Dzung Pham**[3], and **Bennett A. Landman**[4,5,6]

[1]Dept. of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD

[2]National Institute on Aging, National Institute of Health, Baltimore, MD

[3]Dept. of Radiology and Radiological Science, Johns Hopkins University, Baltimore, MD

[4]Dept. of Biomedical Engineering, Johns Hopkins University, Baltimore, MD

[5]Dept. of Electrical Engineering, Vanderbilt University, Nashville, TN

[6]Dept. of Radiology and Radiological Sciences, Vanderbilt University, Nashville, TN

Abstract

Non-invasive neuroimaging techniques enable extraordinarily sensitive and specific *in vivo* study of the structure, functional response and connectivity of biological mechanisms. With these advanced methods comes a heavy reliance on computer-based processing, analysis and interpretation. While the neuroimaging community has produced many excellent academic and commercial tool packages, new tools are often required to interpret new modalities and paradigms. Developing custom tools and ensuring interoperability with existing tools is a significant hurdle. To address these limitations, we present a new framework for algorithm development that implicitly ensures tool interoperability, generates graphical user interfaces, provides advanced batch processing tools, and, most importantly, requires minimal additional programming or computational overhead. Java-based rapid prototyping with this system is an efficient and practical approach to evaluate new algorithms since the proposed system ensures that rapidly constructed prototypes are actually fully-functional processing modules with support for multiple GUI's, a broad range of file formats, and distributed computation. Herein, we demonstrate MRI image processing with the proposed system for cortical surface extraction in large cross-sectional cohorts, provide a system for fully automated diffusion tensor image analysis, and illustrate how the system can be used as a simulation framework for the development of a new image analysis method. The system is released as open source under the Lesser GNU Public License (LGPL) through the Neuroimaging Informatics Tools and Resources Clearinghouse (NITRC).

Keywords

parallel processing; pipeline; rapid prototyping; image processing; MRI

Correspondence to: Bennett A. Landman.

Information Sharing Statement All source code for the JIST infrastructure is available in open source under the Lesser GNU Public License (LGPL) 2.1. Development is hosted by the Neuroimaging Informatics Tools and Resources Clearinghouse (NITRC, http://www.nitrc.org/projects/jist/). Anonymous read access to the source code is enabled, and build instructions can be found on the wiki (http://www.nitrc.org/plugins/mwiki/index.php/jist:AntBuild). Compiled versions of JIST and open source plug-ins are released on this site. Contributed, open source plug-ins are hosted on NITRC under LGPL 2.1 at http://www.nitrc.org/projects/jhumipavplugins/. Other open source and closed source JIST tools are available through the individual author's websites (e.g., http://medic.rad.jhmi.edu/); the license mechanism for third party tools is not restricted by the JIST infrastructure license.

1. Introduction

Medical imaging data can be gathered in such vast quantities that we are often unable to develop and carry out appropriate processing approaches despite publicly available tools that might already exist for carrying out many of the steps. A tool that automatically handles many of the formatting and compatibility issues in order to integrate software from a variety of platforms is needed. The ability to visualize these algorithms through higher level block-diagrams readily revealing inputs and outputs – rather than understanding them through mathematical equations – would be helpful to the entire community and would promote sharing of best practices. While the neuroimaging community has produced many excellent academic and commercial tool packages, new tools are often required to interpret new modalities and paradigms. Yet the development of custom tools and ensuring interoperability with existing tools is a significant hurdle. To address these limitations, we present the *Java Image Science Toolkit (JIST)*, a new framework for algorithm development and large-scale image processing. JIST implicitly ensures interoperability, generates graphical user interfaces, provides advanced batch processing tools, and, most importantly, requires minimal additional programming or computational overhead. With these capabilities, JIST provides an open-source, platform-independent framework to rapidly develop image analysis tools and distribute them to the scientific community.

The neuroimaging community already benefits from several, excellent pipeline environments. These tools facilitate the analysis of large datasets, but are limited in their ability to provide a seamless development path from prototype to cluster-based parallel processing. Existing environments tend to provide end-users with systems to process and visualize results, e.g., the LONI Pipeline (Rex, Ma et al. 2003), NA-MIC Kit (Pieper, Lorensen et al. 2006), FisWidgets (Fissell, Tseytlin et al. 2003), AVS (Sheehan, Fuller et al. 1996), SCIRun (Parker and Johnson 1995), Data Explorer (Lucas, Abram et al. 1992), and Khoros (Konstantinides and Rasure 1994). These end-user systems include graphical user interfaces (GUI's) to facilitate the visual programming in the neuroimaging context (Burnett and McIntyre 1995). The emerging eXtensible Imaging Platform (XIP) is an open source Insight Segmentation and Registration Toolkit (ITK) / Visualization Toolkit (VTK) (Yoo 2004) initiative by caBIG/ DICOM WG23 aimed at providing a visual plug and play programming environment (Mulshine and Baer 2008). Each environment provides specific mechanisms for the incorporation of new image analysis tools as processing modules within the environment. The LONI Pipeline, FisWidgets, and NA-MIC Kit provide manual tools to encapsulate applications, which is advantageous from an end-user standpoint because (almost) any application can be used as a processing module. Alternatively, image analysis tools can be encapsulated with an Application Programming Interface (API) provided by the pipeline environment so that each tool can be automatically recognized as a processing module, as is implemented in SCIrun, Data Explorer, Khoros, and AVS. Developers may prefer the API method of encapsulation because changes to an algorithm's input/output interface are immediately recognized by the pipeline environment; however, this has the disadvantage that tool developers must be aware of the intended pipeline environment.

Although typically considered to be beyond the scope of a "pure" pipeline environment, development of image analysis tools can be expedited by incorporating existing numerical and image analysis libraries, such as ITK (Yoo, Ackerman et al. 2002), VTK (Schroeder, Martin et al. 1996), or MIPAV (McAuliffe, Lalonde et al. 2001). For example, the NA-MIC Kit is greatly enhanced by its close integration with ITK and VTK. However, this is primarily a C/C++/Python solution, so cross-platform compilation and deployment must be carefully managed by a team of experts. The Java programming language (Sun Microsystems, Santa Clara, CA) provides a modern, efficient language which is inherently cross-platform. To date, no pipeline software has made native use of neuroimaging application programming libraries with Java.

We present JIST, which combines a fully functional, graphical pipeline environment with a framework to take advantage of advanced multi-modality imaging libraries available within Java. JIST integrates closely with MIPAV (Medical Image Processing, Analysis and Visualization, National Institutes of Health), a widely used and well-supported multi-dimensional imaging, visualization, and processing package from the National Institutes of Health. Both JIST end-users and programmers can directly incorporate MIPAV functionality. JIST ensures that rapidly constructed prototypes are actually fully-functional processing modules with support for multiple GUI's, a broad range of file formats, and distributed computing features. JIST enables developers to focus on implementing the innovative aspects of their algorithm instead of re-implementing common functionality.

JIST combines a solution for the interactive processing of individual datasets with an efficient large-scale batch processing infrastructure. The close integration of the JIST framework with MIPAV provides for user-friendly visualization and exploration of the multi-dimensional imaging data as well as three-dimensional structures. MIPAV includes triplanar, volumetric, mesh surface, and stream-tube visualization tools in a cross-platform application. Therefore, JIST may be used to encapsulate difficult and/or time consuming steps in the analysis and visualization of data from a single subject, or the framework may be used to process data from many subjects where time affords less opportunity to inspect/optimize each processing step. Furthermore, MIPAV includes sophisticated labeling and interactive region of interest analysis frameworks for both multi-dimensional imaging data and surface meshes. These tools may be used to develop semi-automated or supervised image analysis routines, which may exploit human interaction with functionality in the JIST framework. Finally, JIST and MIPAV support standard file formats, so that analysis results may be rendered with general purpose tools, such as ParaView (Kitware, Inc., Clifton Park, NY) or Amira (Visage Imaging, Inc. San Diego, CA).

This manuscript is organized as follows. Section 2 presents an overview of the JIST software architecture. Section 3 presents three case studies for the JIST system: as a platform for (1) cortical surface analysis of the human brain, (2) diffusion tensor imaging of brain connectivity, and (3) as a simulation environment for development of a new imaging method. Finally, section 4 discusses potential implications of this work.

2. Software Architecture

The JIST toolkit is written entirely in Java and distributed as a platform independent Java Archive (JAR) file which can be executed by any Java enabled platform. Full support is provided for the Sun Java version 1.6 and above. JIST development is hosted by the Neuroimaging Informatics Tools and Resources Clearinghouse (NITRC) (Kennedy 2007) and distributed as an open-source Java project licensed under the Lesser GNU Public License 2.1 (Foundation 2007).

The JIST API extends MIPAV's image analysis API and plug-in framework and provides a modular infrastructure for developing tools with several programmatically generated interfaces. These interfaces are based on the specified input/output parameters for each tool, so the programmer need only be concerned with the type of information being passed rather than how an application might load or save the data. Processing algorithms are automatically recognized by JIST plug-ins and can use any functionality contained in the image analysis API. Since the JIST API extends the native MIPAV interface, numerous libraries for file format handling (over 70 image formats), image algebra (for 2-, 3-, and 4- dimensional images), registration (both linear and nonlinear methods), segmentation (including Markov random fields, topological consistency, level sets, atlas based methods), and other common functionalities (e.g., skull removal, inhomogeniety correction, diffusion tensor analysis, etc.) are available. Further detail is given in section 2.1.

In addition to the JIST API, the JIST framework consists of three key graphical tools which allow end-users to interact with the system (detailed descriptions are below):

- The *Plug-In Selector* (Figure 1) is a MIPAV plug-in that provides a GUI for the user to select and execute JIST tools as (regular) MIPAV plug-ins. JIST processing algorithms are automatically discovered by the *Plug-In Selector*, so there is no need to individually install processing algorithms.
- The *Pipeline Layout Tool* (Figure 2) is a visual editor for designing processing pipelines. Users can design pipelines to run with different sets and ranges of parameters. The *Pipeline Layout Tool* automatically detects image analysis algorithms developed within the JIST framework so that they can be seamlessly incorporated into pipelines. Native MIPAV functionality can be capture with simple adapter class files.
- The *Process Manager* (Figure 3) manages execution of processing tasks in a multi-processor computing environment or through the Distributed Resource Management Application API (DRMAA) (Rajic, Brobst et al. 2004) which supports processing grids. During execution, the *Process Manager* collects information about the speed and memory performance of each algorithm in addition to any debugging information. Experimental results are then deposited into spreadsheets or filed into directories.

The graphical programs are encapsulated as MIPAV plug-ins so that they can be accessed from the MIPAV plug-in interface; therefore, these modules and the processing algorithms can be installed through MIPAV's plug-in installation tool. Alternatively, the programs may be started and used as independent applications and do not require an open instance of the MIPAV user interface. Additionally, JIST easily integrates with Matlab (Mathworks, Natick, MA) since Java objects can be directly accessed from the Matlab command line. Thus, JIST functionality can be used and/or tested within the Matlab environment.

2.1 Application Programming Interface (API)

Developers can use the API for its image analysis functionality and then encapsulate their algorithm in a JIST processing algorithm so that it will be automatically recognized by the pipeline environment. JIST processing algorithms adhere to a template design pattern. Developers are responsible for implementing three methods that are called within the processing algorithm's execution cycle. Two of these methods (createInputParameters and createOutputParameters) are responsible for appending algorithm specific parameter objects to the collection of input or output parameters. The third method (execute) is responsible for calling the image analysis algorithm using parameter values stored in the input parameters and placing results from the image analysis algorithm into the output parameters. The process algorithm infrastructure handles the remaining execution steps generically for all image analysis algorithms.

GUIs are automatically generated based on input/output parameters specified within each processing algorithm. JIST supports parameter objects for integers, floats, booleans, enumerations, files, file collections, and more specific parameter types for images (2D/3D/4D) and surfaces. Restrictions can be placed on parameter objects to limit the range or type of values accepted by a parameter. GUI components generated for each parameter type have a consistent look-and-feel, but developers have the option to use a custom GUI component to render a parameter.

Since JIST is an open source development platform, programmers can extend the JIST API to incorporate new input/output parameter types and customize user interfaces for particular applications. With these capabilities, JIST's highly object-oriented framework provides a modular and flexible approach to image analysis software development that encourages reuse and extension of existing software.

2.2 Graphical Tools

2.2.1 JIST Plug-In Selector (Figure 1)—The JIST API provides a standard look-and-feel for all plug-ins and options for opening/saving input parameters (Figure 1). Every JIST plug-in has a file menu for Open Algorithm Input, Save As Algorithm Input, and Save As Module Definition. Selecting Save As Algorithm Input will save an XML description of all the current input parameters, which can be loaded back into the plug-in by selecting Open Algorithm Input. Selecting Save As Module Definition allows users to overwrite or create a new instance of a plug-in from the library that uses the current input parameters as defaults. All plug-ins have an input parameter for Algorithm Information. The information fields may contain additional documentation, author lists, contact information, relevant citations, and links to online resources. This information is mined from the algorithm's source code and XML module description. Once all input parameters are specified, users can click Ok to start the algorithm. If a parameter value is invalid, an error will be displayed; otherwise, the user will be prompted to select a directory to save the output from the algorithm. Clicking Cancel will cancel execution of the plug-in. Once an algorithm has finished, another dialog displays a summary of all the output parameters including information about execution time. All output information is saved in the directory that was previously specified, including an XML description of the input and output parameters.

2.2.2. *Pipeline Layout Tool* (Figure 2)

2.2.2.1 Layout Panel: The layout panel provides a graphical interface to arrange pipeline modules (Figure 2a). Modules can be dragged and dropped from the module panel into the layout panel. All algorithm modules have input and output ports represented by circles and triangles respectively. Ports that have been assigned valid values are represented by filled circles. If the user clicks and holds down on a port, ports that are compatible with this port will appear green. Cycles in the dependency graph are detected and prevented; connections that would be compatible but would introduce cycles do not turn green. Users can then drag the cursor to a green port to form a connection between ports represented by a connector edge. Some output ports are representative of a list of values. In which case, the connector will have a number next to it representing an index into the list of values. The index number can be edited by double-clicking on the index, which will bring up a spinner box. The name of modules can be edited by double-clicking on the name of the module. Module names must be unique. If the user specifies a name that already exists, a number will be appended to the name. Algorithm modules can be grouped together by selecting multiple modules and clicking the group button in the toolbar. After which, a box outline will appear around the collection of modules. Groups can be collapsed/expanded by clicking the −/+ button in the top left-hand corner of the group or by clicking the collapse/expand button in the toolbar. Algorithm modules and groups can be saved as module definitions by right clicking on the module or group and selecting Save As Module Definition. After which, the module panel will be reinitialized.

2.2.2.2 Parameter Panel: When a module is selected in the layout panel (Figure 2b), a dialog appears in the parameter panel that allows the user to edit input parameters for that module. This dialog looks identical to the plug-in dialog, with the exception of a toggle button next to each parameter value. The toggle button indicates whether this parameter prefers to use a value specified from another module. When the connection button is toggled on, an option box will appear that allows the user to select an output port from another module to use as input to the current input port. Selecting an output port will create a connector between ports in the layout. If the output port has a collection of values, then a spinner box will appear next to the connection name that allows the user to specify an index into the array of outputs.

2.2.2.3 Module Library: Within the module library (Figure 2c), algorithm modules can be edited by right-clicking on a module and selecting Edit or Delete from the pop-up menu. Algorithm modules can also be dragged and dropped within the module panel to change the file structure of module definitions. New folders can be added to the library by right-clicking and selecting New from the pop-up menu.

There are many ways to specify multiple inputs to an algorithm. A source collection allows users to create a list of parameter values. A source list allows users to specify a file that contains a listing of parameter values. A source directory allows users to specify a directory that contains files to use as inputs, and a source sweep allows users to specify a range of numerical or enumeration values to iterate through. Sources can be nested to form foreach loops by connecting the triangle on the right side of each source module with the circle on the left side of another source module.

Destination modules manipulate output from all executions of particular processing algorithms. The Summary destination records a string representation of a parameter for all experiments. The Copy Files destination copies all selected output parameter files to a specified directory. The Assertion Test destination generates a spreadsheet containing a list of all experiments, the source settings for each experiment, and a boolean value indicating whether the assertion test passed or not for each experiment.

2.2.3 Process Manager (Figure 3)—The JIST *Process Manager* permits real-time monitoring and managing of processing tasks (Figure 3). JIST automatically resolves dependencies between processing tasks so that tasks are executed in the appropriate order. When a layout file is loaded, a directed graph of algorithm dependency is constructed for each experiment — an experiment in JIST consists of a unique set of inputs from the layout sources. The list of all module-experiment pairs (i.e., tasks) is listed in the status table (Figure 3b). If the dependencies are not met, then a task is listed as NOT READY. If a task has previously failed or the results on the disk are not consistent with the current task state, then the task is listed as FAILED or OUT-OF-SYNC, respectively. Otherwise, the task is listed as READY and the task may be started. Running tasks are listed as RUNNING until the process has finished or is canceled. The input dependencies for each task can be viewed by selecting a line and viewing the ancestor pane (Figure 3c), while the output data and tasks dependent upon a task are listed in the result pane (Figure 3d).

In the event that more than one task can be run simultaneously, the *Process Manager* attempts to distribute tasks across multiple processors and displays real-time information about process status, computation time, memory usage, algorithm progress, task dependencies, and algorithm arguments. A task's priority can be modified, and the user can select a subset of tasks to run, stop, clean, or rerun. Debugging information generated by the *STDOUT* and *STDERR* streams is also captured and available for inspection. If a processing grid is available, users can choose to forward tasks to the processing grid instead of running them on their local workstation. The *Process Manager* is DRMAA compliant, which permits the JIST pipeline environment to be used for large scale processing tasks that require more resources than locally available. The DRMAA engine handles task scheduling and resource assignment, while JIST validates successful completion of each task and prevents resource allocation to tasks for which dependent steps have not finished or have reported critical errors.

2.3 Testing Framework

All image analysis algorithms developed within the JIST framework are testable through the *Pipeline Layout Tool*. Users can design layouts that execute image analysis algorithms with a variety of different input settings. The output of each algorithm can be forwarded to a module that compares the result against an existing result and generates a boolean assertion indicating whether the algorithm passed or failed the test. Destination modules can be used to collect information about each test case, including its test case identifier, input description, output description, and a boolean expression indicating whether the test passed or not. In addition to automated unit testing, the *Pipeline Layout Tool* can be used for automated integration and system testing by concatenating multiple image analysis algorithms into a pipeline. Developers can choose to distribute the pipeline as a layout file, or consolidate algorithm calls into a single processing module. Furthermore, test-driven development is expedited by the Process Manger's ability to parallelize test case execution across multiple processors and multiple computers.

2.4 Command Line Interfaces

All JIST modules and layouts may be run directly from the command line. Calling syntax is programmatically generated and parsed for both required and optional inputs using standard Portable Operating System Interface (POSIX) syntax. A human readable, structured self-description of each module or layout may be retrieved from the command line which may be used to embed JIST functionality in other programs, pipelines, or automated testing environments. Alternatively, this structured text may be parsed by scripts to automatically link JIST into an existing infrastructure.

JIST supports integration of tools that do not implement the JIST API through a slightly restricted version of the LONI xml description for command line modules. If a tool is written in Java, such as the CAMINO diffusion weighted MRI toolkit (Cook, Bai et al. 2006), it is often a trivial matter to write an adapter layer so that the functionality is available natively within MIPAV/JIST. With the permission of the authors, several diffusion tensor fitting routines from the CAMINO were incorporated into JIST. For complex, platform dependent software packages (i.e., the FMRIB Software Library – FSL (Smith, Jenkinson et al. 2004)), it would be a simpler matter to use an xml description to encapsulate the functionality for use in JIST.

3 Case Studies

JIST provides an ideal environment for developing and publishing software for medical image analysis. In a typical scientific setting, research is focused on improving algorithms and enabling new inferences. Clearly, software capabilities, interoperability, and user support are important, but investigators are rarely well-funded to provide “turn-key” solutions for end-users. As illustrated in the subsequent case studies of (1) cortical surface estimation, (2) diffusion tensor imaging, and (3) simulation of model fitting, JIST streamlines implementation, testing, and analysis of neuroimaging data and provides for a direct and efficient mechanism for publishing software. The capabilities of JIST that are most crucial to these efforts are:

- **Cross-platform support** – Investigators can rarely dictate the operating system of their collaborators and end-users, and developing multi-platform source code can be exceptionally challenging. The following case studies rely on well-tested

MIPAV infrastructure for all common platforms, including Windows, Macintosh, Unix, and other Java-enabled systems.

- **File format support** – Standardized file formats (e.g., the Neuroimaging Informatics Technology Initiative, NIfTI, file format) are important, yet there are scores of competing and legacy formats. Full support for even the “common” formats is a major undertaking. The following case studies rely on MIPAV's extensive imaging format support for the majority of the data and implement simple, specialized format tools for the necessary proprietary file formats (e.g., the fiber data format in the diffusion tensor imaging example).
- **Extensive Library of Image Analysis Routines** — Implementing well-understood tasks, such as affine image registration, are straight forward, but can be incredibly time-consuming. JIST accelerates development by providing direct access to the MIPAV API, numerical and optimization libraries, and hundreds of publicly available image processing modules.
- **Modular Algorithm Structure** — In JIST, complex algorithms are constructed by concatenating a series of simpler steps. These individual steps can be used as originally designed or reused in new ways as the non-rigid registration routine was reused from the cortical surface example in the diffusion tensor case study.
- **Scalable / Batch Processing** — For clinical case studies and single subject analyses, it is very important to be able to process a single subject at a time in an efficient manner and to manually inspect each analysis step, while for large cross-sectional and/or longitudinal studies, automated processing capabilities are essential. JIST provides plug-in support so that all of the tools described may be run easily on single datasets and a process manager to handle batch processing for large studies.
- **Parameter Sweeps and Testing** — Development of new algorithms often involves extensive parameter optimization and testing under various conditions. JIST data sources can be simply nested in a hierarchical “for each” combination so that complex parameter spaces can be examined simply by connecting pins. The simulation framework in the model fitting case study can combine this functionality with the ability of modules to both synthesize and analyze data.
- **Visualization** — The ability to visualize and interact with results is critical to ensuring that tools are properly functioning and data are properly analyzed. JIST's close integration with MIPAV provides a robust platform for visualization of both final intermediate results from multi-dimensional imaging studies. Additionally, JIST supports common raster and vector graphic formats to that data may be readily exported to generic visualization tools.
- **Publishing** — Publishing usable, quality software is dependent upon being able to provide sufficient support for end-users for the complete analysis framework (from installation to analysis and visualization). JIST greatly reduces this burden as software installation, file formats, and visualization are supported by the NIH MIPAV development team and the infrastructure is supported by the JIST authors through NITRC. To provide software support within the JIST framework, authors need only to release (1) a JAR containing the modules unique to their analyses, (2) a “.layout” file, and (3) a tutorial on how to use and interpret the results.

These diverse imaging tasks are representative of the multi-step, often iterative processing techniques that form the core of magnetic resonance imaging groups. All of these experiments could run on simple hardware (such as a notebook computer) or large compute clusters. For these case studies, we used an eight core (2.92 GHz) Linux workstation with 32 GB of RAM.

3.1 Cortical Surface Estimation

Cortical Reconstruction Using Implicit Surface Evolution (CRUISE) was developed as a computationally efficient way to robustly identify the cortical surface boundaries (i.e., the surfaces between the outer layer of cortical gray matter and the non-brain tissue and the layer between the cortical gray matter and the brain's inner white matter) (Han, Pham et al. 2004). The initially published CRUISE software consisted of numerous C programs that were concatenated together using a typical shell-script pipeline. Although CRUISE was shared with collaborators and used in clinical research, the full package could not be released to the public due to the burden that supporting the software would place on the authors.

In the JIST implementation, each C program was ported into a JIST module written in Java. The porting process was accelerated by utilizing MIPAV's image analysis functionality instead of re-implementing functionality in the custom C libraries. Each processing module was then unit tested by constructing a pipeline that compared the result from the processing module against a previous result computed with the C version. Spreadsheets were generated to indicate whether output images were the same or not, within a certain numerical tolerance. After each processing algorithm had been independently tested, modules were incrementally concatenated and tested to make sure the resulting pipeline was a faithful reproduction of the original CRUISE pipeline. With the JIST layout, individual steps could be easily developed, optimized, and integrated without hacking shell scripts. Since the time of the initial Java port, a topologically consistent tissue classification algorithm (TOADS) has been added to CRUISE, replacing several original steps and producing a more robust pipeline (Bazin and Pham 2006).

As a demonstration of the JIST-enabled capabilities, the CRUISE pipeline was run on 147 OASIS datasets (Marcus, Wang et al. 2007). Thickness, curvedness, and shape index were computed on the central surface (Tosun, Rettmann et al. 2004; Tosun, Rettmann et al. 2004). Figure 4 depicts the processing pipeline (a), central surface for an individual subject (b), and median surface measurements displayed on an inflated atlas surface (c). The entire study, including CRUISE preprocessing and post-processing, took approximately 4 days 4 hours real time. This run used 26 days 3.5 hours of CPU time running on 6 cores.

3.2 Diffusion Tensor Imaging

Diffusion tensor imaging is widely used to study structural connectivity patterns and the integrity of white matter (the "information highway" of the nervous system) (Basser and Jones 2002). Diffusion-inferred connectivity and integrity metrics are computed for a series of sensitized three-dimensional MRI images (typically 30–90 volumes). Processing of this data must take into account potential subject motion, scanner induced geometric distortions, and the relation between the sensitization and geometries. To automate this procedure, we developed CATNAP (Coregistration, Adjustment, and Tensor-solving – a Nicely Automated Program) (Landman, Farrell et al. 2007) and released this system for fully automated processing. Since a Matlab implementation of fiber tracking was too slow for routine use, computation of fiber paths from the data required semi-automated processing with Fiber Assignment by Continuous Tracking (FACT, (Mori, Crain et al. 1999)) in MRIStudio (http://www.mristudio.org). Providing support for the manufacturer specific file formats became untenable for the developers while maintaining a research program, so we chose to port CATNAP and FACT to a JIST API to take advantage of the extensive file format support provided by MIPAV.

In the port of CATNAP, each functional Matlab routine was ported to a JIST module. The FACT algorithm was also implemented to enable end-to-end, fully-automated processing. Additionally, the registration tools previously implemented for CRUISE enabled improved distortion correction and superior integration with cortical surfaces. Figure 5 depicts a cortical reconstruction with fiber tracks obtained using JIST. The real time to process the high resolution DTI data set was 55 minutes. This run used 2 hours 14 minutes of CPU time running on 6 cores. Note full utilization of the parallel processing bandwidth was not possible since only a single dataset was analyzed and there were substantial dependencies.

3.3 Simulation and Optimization Framework

Algorithm development requires extensive testing, validation, and optimization. In light of the numerous experiments used to optimize and evaluate performance, it can be quite a task to setup experiments and record how each experiment was run. Although the JIST infrastructure was principally designed for data analysis, modules can equivalently provide a framework for simulation. The parameters for simulation modules may be controlled by data sources, which can provide numeric and text inputs in addition to imaging/surface data. The *Pipeline Layout Tool* provides functionality to create nested sources (i.e., “for each”) by connecting source pins and to explore sequential sets of inputs (i.e., arithmetic series, arbitrary lists, etc.). Therefore, parameter spaces can be easily explored. JIST records the complete input parameters for each task in the output directories in a text-based XML format so that the precise module versions and parameters may be recovered using any text editor in the case that the JIST layout control file is misplaced or modified.

In this case study, we demonstrate how JIST can be used to produce simulated data. This simulation is then used to evaluate and optimize the performance of a new analysis routine. Diffusion tensor imaging suffers serious limitations in regions of crossing fibers because traditional tensor techniques cannot represent multiple, independent intra-voxel orientations. The authors have recently proposed compressed sensing as a method to resolve crossing fibers using a tensor mixture model (e.g., Crossing Fiber Angular Resolution of Intra-voxel structure, CFARI) (Landman, Bogovic et al. 2008). Although similar in spirit to deconvolution approaches, CFARI uses sparsity to stabilize estimation with limited data rather than spatial consistency or limited model order. CFARI was natively developed, optimized, and validated within the JIST framework with a direct connection to Matlab. JIST provided an ideal framework for parameter tuning and optimization. JIST modules were developed to create data models, simulate data, initialize reconstruction parameters, and estimate underlying structure as illustrated in Figure 6. The real time and CPU time to process the simulation were less than a minute using one CPU core.

4 Discussion and Conclusion

New types of medical imaging data – alternate contrasts, resolution, noise levels, etc. – arise constantly due to hardware improvements or innovations in imaging methodology. It is vital to be able to test existing algorithms to see whether processing capability already exists to carry out quantitative tasks with these new data. Many medical imaging analysis processes involve the concatenation of many steps; this is especially true in neuroimage analysis. Often, these so-called processing pipelines are carried out using scripts, typically within the Unix environment. Modification and refinement of such scripts for new data or alternate parameters or algorithms must typically be carried out by a programmer in order to avoid making mistakes and implementing a “wrong” pipeline. It is highly desirable to have a graphical environment that has internal consistency checks for the validity of a processing pipeline and that clearly presents what steps are actually being carried out. JIST provides this capability in the context of a rapid prototyping environment and enables testing of different algorithms and of different parameters for these algorithms. JIST's *Process*

Manager scales from single-core notebook computers to multi-processor servers and multi-node grid computing environments. Hence, JIST enables researchers to make rapid progress in the use of new data for scientific research or clinical research explorations.

Table 1 shows a feature comparison of visualization, rapid prototyping, and pipelining environments that are frequently used or currently being developed for medical imaging. AVS, SCIRun, and OpenDX are primarily oriented towards modeling and visualization of general multidimensional data. As such, they lack many algorithms specifically designed for medical image analysis. FisWidgets provides a menu-driven graphical user interface (GUI) to multiple neuroimaging analysis algorithms and through its GUI, allows construction of sequences of algorithms and loops. The NA-MIC kit includes image analysis algorithm libraries, a visualization and analysis tool (3-D Slicer), and a scripting tool for creating batch processes (BatchMake). The packages most comparable to JIST are the LONI Pipeline and XIP. Each package is freely available, and utilizes a visual programming interface to construct data workflows by forming connections between processing modules. Although there is overlap in functionality in all these approaches, there are differences between the packages in the licensing, implementation, and application focus. The LONI Pipeline supports a variety of image analysis operations and grid computing capabilities. However, it does not currently distribute its source code and does not support an image analysis API. XIP is an ambitious package, currently in beta release, that seeks to provide both extensive visualization and data processing features, as well as clinical standards compliance in supporting DICOM WG23 interfaces. Although new modules can be built within XIP, it is currently not straightforward to integrate existing executable applications.

The software tools described in Table 1 have been designed to balance different tradeoffs and different user communities will find these tradeoffs more or less acceptable. For example, some tools provide improved support for near real-time interaction (e.g., OpenDX) versus batch processing (LONI, JIST), or for using a cross-platform language (e.g., LONI, JIST) versus managing a multi-platform build (e.g., XIP, NAMIC). As described in Section 3, JIST achieves a favorable balance between enabling simple, exploratory development of new algorithms while providing end users with a robust set of tools. A common criticism is that Java runs within a virtual environment and does not make native use of hardware resources, so it cannot be as fast as other compiled languages. However, modern Java compilers achieve within 20 percent the performance on traditional computational benchmarks of C/C++ and FORTRAN compilers (and Java routines have, in some instances, been shown to be faster than C compiled code) (Boisvert, Moriera et al. 2001;Bull, Smith et al. 2001). Furthermore, libraries are available based on BLAS and LAPACK to allow Java routines to make use of machine-optimized numeric libraries (e.g., matrix-tools-java, http://code.google.com/p/matrix-toolkits-java/). Clearly, the authors advocate that JIST is a practical and desirable solution for many in the medical imaging community, but it is not the only solution. The authors support and encourage integration of the JIST framework and JIST modules into other packages, such as XIP, the NA-MIC kit and the LONI Pipeline, via command line interfaces or using network interface layers. Integration with other platforms is an active area of research.

JIST combines the highly desirable capabilities of existing image analysis tools for rapid prototyping and cross-platform software release into a comprehensive package for software development in Java. Specifically, JIST's MIPAV plug-in framework enables rapid development of portable image analysis tools in which all plug-ins are dressed with an auto-generated GUI for improved usability and a consistent look-and-feel. JIST encapsulates image analysis algorithms as processing modules that have compatible interfaces so that they can be concatenated together to form pipelines. Users can then construct test cases by adding sources and destinations to their pipelines. The automated testing framework encourages test-driven development to improve the robustness of image analysis tools. Furthermore, the API's highly object-oriented framework encourages development of extendable and reusable code that is easier to maintain. These capabilities permit developers to focus on implementing the innovative aspects of their algorithm instead of re-implementing common functionality that is desirable for all image analysis tools, thereby providing software that better serves the image analysis and neuroimaging community.

Acknowledgments

We greatly appreciate the unwavering support of Matt McAuliffe and Evan McCreedy of the NIH Center for Information Technology and the dedication of our undergraduate interns (Yufeng Guo, Robert Kim, Meenal Patel, Heba Mustufa, Hanlin Wan, and Jie Zhang). This work was supported by NIH/NINDS 5R01NS037747 and 1R01NS056307 and NIH/NIA N01-AG-4-0012.

5 References

Basser PJ, Jones DK. Diffusion-tensor MRI: theory, experimental design and data analysis - a technical review. NMR Biomed 2002;15(7–8):456–67. [PubMed: 12489095]

Bazin, P.; Pham, D. TOADS: topology-preserving, anatomy-driven segmentation. Biomedical Imaging: Macro to Nano, 2006. 3rd IEEE International Symposium on; 2006.

Boisvert RF, Moriera J, et al. Java and Numerical Computing. Computing in Science & Engineering 2001;3(2):18–24.

Bull JM, Smith LA, et al. Benchmarking Java against C and Fortran for Scientific Applications. EPCC. 2001

Burnett M, McIntyre D. Visual Programming. Computer 1995:14–16.

Cook, PA.; Bai, Y., et al. Camino: Open-Source Diffusion-MRI Reconstruction and Processing. 14th Scientific Meeting of the International Society for Magnetic Resonance in Medicine; Seattle, WA, USA. 2006.

Fissell K, Tseytlin E, et al. Fiswidgets. NeuroInformatics 2003;1(1):111–125. [PubMed: 15055396]

Foundation, FS. Lesser General Public License. 2007. from http://www.gnu.org/licenses/lgpl-2.1.txt

Han X, Pham D, et al. CRUISE: Cortical reconstruction using implicit surface evolution. NeuroImage 2004;23(3):997–1012. [PubMed: 15528100]

Kennedy D. Neuroinformatics and the Society for Neuroscience. NeuroInformatics 2007;5(3):141–142.

Konstantinides K, Rasure J. The Khoros software development environment for image and signalprocessing. Image Processing, IEEE Transactions on 1994;3(3):243–252.

Landman, B.; Bogovic, J., et al. Compressed sensing of multiple intra-voxel orientations with traditional DTI. Proc Workshop on Computational Diffussion MRI, MICCAI 2008; 2008.

Landman B, Farrell J, et al. Effects of diffusion weighting schemes on the reproducibility of DTI-derived fractional anisotropy, mean diffusivity, and principal eigenvector measurements at 1.5 T. NeuroImage 2007;36(4):1123–1138. [PubMed: 17532649]

Lucas B, Abram G, et al. An architecture for a scientific visualization system. 1992

Marcus D, Wang T, et al. Open Access Series of Imaging Studies (OASIS): cross-sectional MRI data in young, middle aged, nondemented, and demented older adults. Journal of Cognitive Neuroscience 2007;19(9):1498–1507. [PubMed: 17714011]

McAuliffe, M.; Lalonde, F., et al. Medical image processing, analysis and visualization in clinical research. Computer-Based Medical Systems, 2001. 14th IEEE Symposium on; 2001.

Mori S, Crain B, et al. Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging. Annals of Neurology 1999;45(2)

Mulshine, JL.; Baer, TM. Quantitative Imaging Tools for Lung Cancer Drug Assessment. Wiley-OSA; 2008.

Parker, S.; Johnson, C. SCIRun: a scientific programming environment for computational steering. ACM New York, NY, USA: 1995.

Pieper, S.; Lorensen, B., et al. The NA-MIC Kit: ITK, VTK, pipelines, grids and 3D slicer as an open platform for the medical image computing community. Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on; 2006.

Rajic, H.; Brobst, R., et al. Distributed Resource Management Application API Specification 1.0. 2004. from http://www.ggf.org/documents/GWD-R/GFD-R.022.pdf

Rex DE, Ma JQ, et al. The LONI Pipeline Processing Environment. NeuroImage 2003;19(3):1033–1048. [PubMed: 12880830]

Schroeder, W.; Martin, K., et al. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. Visualization, 1996. 7th IEEE Conference on; 1996.

Sheehan B, Fuller S, et al. AVS software for visualization in molecular microscopy. Journal of Structural Biology 1996;116(1):99–106. [PubMed: 8742730]

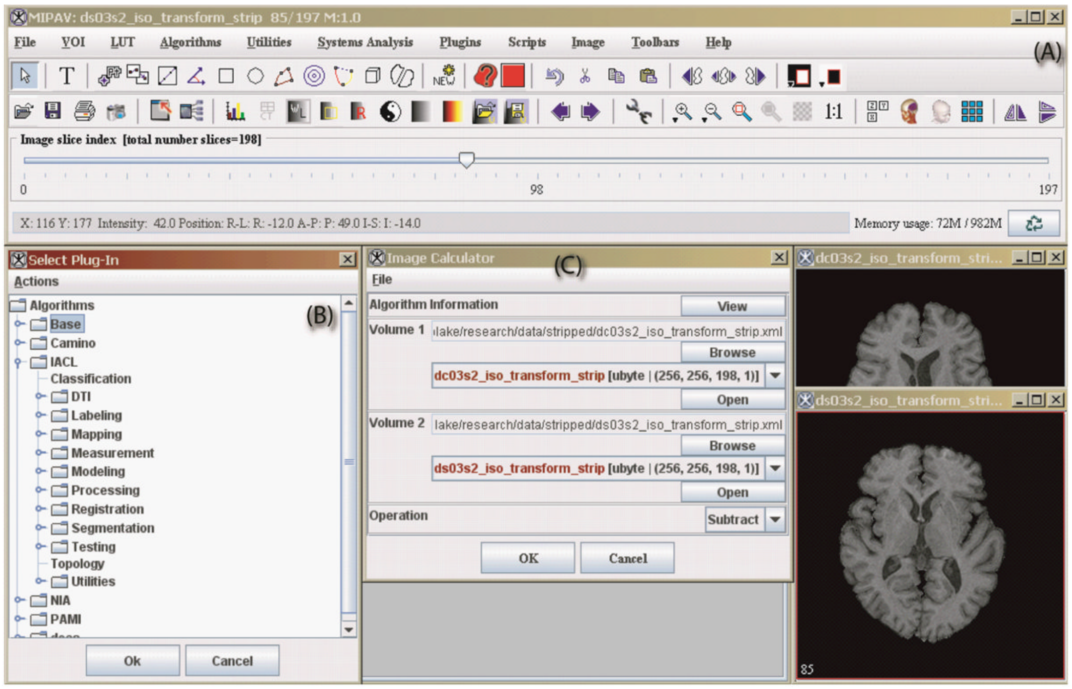
Smith SM, Jenkinson M, et al. Advances in functional and structural MR image analysis and implementation as FSL. Neuroimage 2004;23(Suppl 1):S208–19. [PubMed: 15501092]

Tosun D, Rettmann M, et al. Cortical Reconstruction Using Implicit Surface Evolution: A Landmark Validation Study. Lecture Notes in Computer Science 2004:384–392.

Tosun D, Rettmann M, et al. Mapping techniques for aligning sulci across multiple brains. Medical Image Analysis 2004;8(3):295–309. [PubMed: 15450224]

Yoo, T. Insight into Images. A.K. Peters; 2004.

Yoo T, Ackerman M, et al. Engineering and Algorithm Design for an Image Processing API: A Technical Report on ITK-the Insight Toolkit. Studies Health Technology and Informatics 2002;85:586–592.



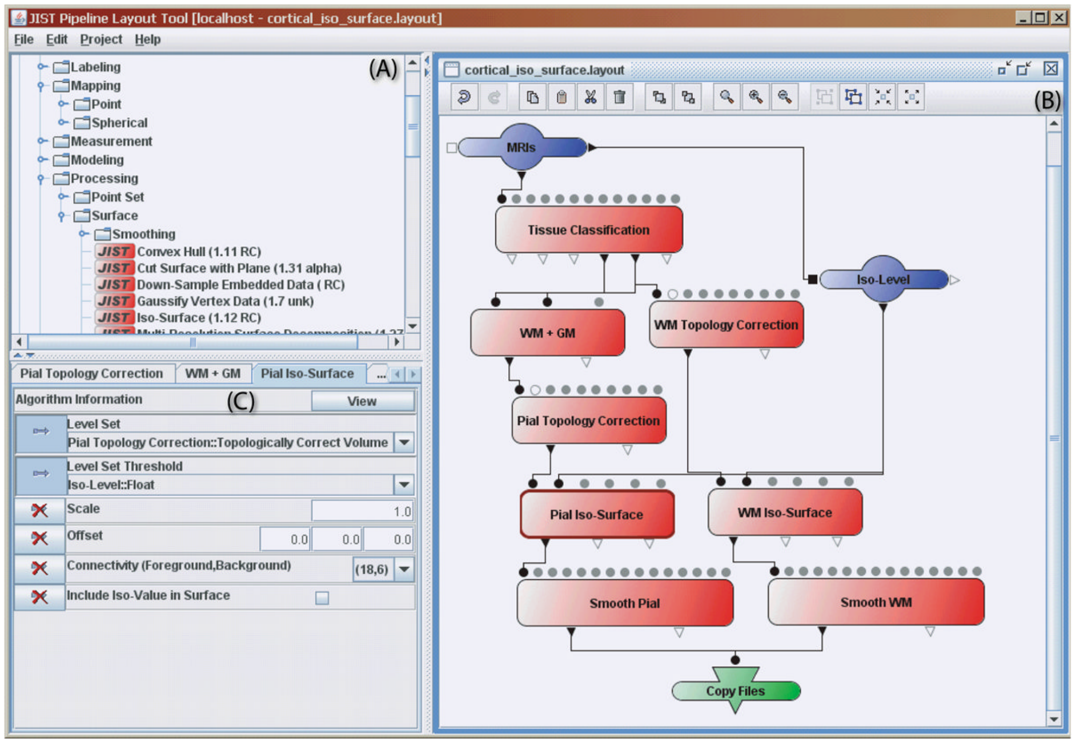
X: 116 Y: 177 Intensity: 42.0 Position: R-L: R: -12.0 A-P: P: 49.0 I-S: I: -14.0

Volume 1 lake/research/data/stripped/dc03s2_iso_transform_strip.xml
dc03s2_iso_transform_strip [ubyte | (256, 256, 198, 1)]

Volume 2 lake/research/data/stripped/ds03s2_iso_transform_strip.xml
ds03s2_iso_transform_strip [ubyte | (256, 256, 198, 1)]

Operation Subtract

Figure 1.
MIPAV (A) automatically detects the presence of the *Plug-In Selector* and provides a menu item to open the tool. The *Plug-In Selector* interface lists all detected plug-ins according to their programmer specified hierarchy (B). Once selected from this menu, a programmatically generated GUI appears for any tool (C) which can operate on any image accessible to MIPAV.



Level Set
Pial Topology Correction::Topologically Correct Volume

Level Set Threshold
Iso-Level::Float

Scale 1.0

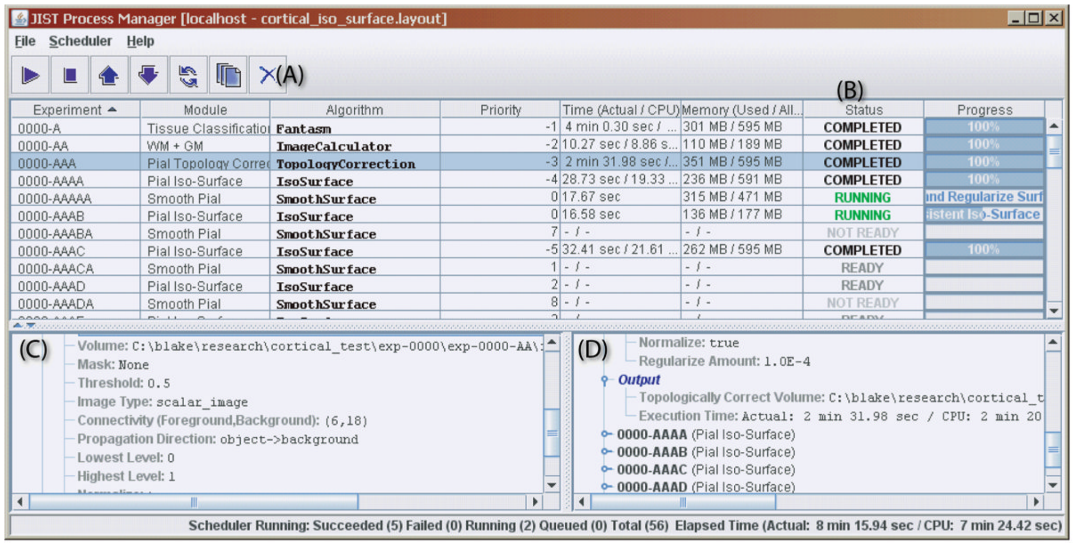
Offset 0.0 0.0 0.0

Connectivity (Foreground,Background) (18,6)

Include Iso-Value in Surface ☐

Figure 2.

The *Pipeline Layout Tool* is accessible as a MIPAV plug-in or as an independent program. This interface detects and lists all available analysis tools and input/output interfaces on the left (A). These modules may be dragged into the visual programming interface on the right (B). When a particular module is selected, the options for that module appear in the parameter panel (C). These options may be manually adjusted or connected to “pins” on other modules to enable information flow. Cyclic loops are detected and disallowed.


JIST Process Manager [localhost - cortical_iso_surface.layout]
File
Scheduler
Help
(A)
(B)
(C)
(D)

Experiment	Module	Algorithm	Priority	Time (Actual / CPU)	Memory (Used / All...	Status	Progress
0000-A	Tissue Classificatio	**Fantasm**	-1	4 min 0.30 sec / ...	301 MB / 595 MB	COMPLETED	100%
0000-AA	WM + GM	**ImageCalculator**	-2	10.27 sec / 8.86 s...	110 MB / 189 MB	COMPLETED	100%
0000-AAA	Pial Topology Correc	**TopologyCorrection**	-3	2 min 31.98 sec /...	351 MB / 595 MB	COMPLETED	100%
0000-AAAA	Pial Iso-Surface	**IsoSurface**	-4	28.73 sec / 19.33 ...	236 MB / 591 MB	COMPLETED	100%
0000-AAAAA	Smooth Pial	**SmoothSurface**	0	17.67 sec	315 MB / 471 MB	RUNNING	ınd Regularize Surf
0000-AAAB	Pial Iso-Surface	**IsoSurface**	0	16.58 sec	136 MB / 177 MB	RUNNING	istent Iso-Surface
0000-AAABA	Smooth Pial	**SmoothSurface**	7	- / -	- / -	NOT READY	
0000-AAAC	Pial Iso-Surface	**IsoSurface**	-5	32.41 sec / 21.61 ...	262 MB / 595 MB	COMPLETED	100%
0000-AAACA	Smooth Pial	**SmoothSurface**	1	- / -	- / -	READY	
0000-AAAD	Pial Iso-Surface	**IsoSurface**	2	- / -	- / -	READY	
0000-AAADA	Smooth Pial	**SmoothSurface**	8	- / -	- / -	NOT READY	

(C)
- Volume: `C:\blake\research\cortical_test\exp-0000\exp-0000-AA\`
- Mask: `None`
- Threshold: `0.5`
- Image Type: `scalar_image`
- Connectivity (Foreground,Background): `(6,18)`
- Propagation Direction: `object->background`
- Lowest Level: `0`
- Highest Level: `1`

(D)
- Normalize: `true`
- Regularize Amount: `1.0E-4`
- *Output*
  - Topologically Correct Volume: `C:\blake\research\cortical_t`
  - Execution Time: `Actual: 2 min 31.98 sec / CPU: 2 min 20`
- **0000-AAAA** (Pial Iso-Surface)
- **0000-AAAB** (Pial Iso-Surface)
- **0000-AAAC** (Pial Iso-Surface)
- **0000-AAAD** (Pial Iso-Surface)

Scheduler Running: Succeeded (5) Failed (0) Running (2) Queued (0) Total (56) Elapsed Time (Actual: 8 min 15.94 sec / CPU: 7 min 24.42 sec)

Figure 3.

The *Process Manager* may be started as a MIPAV plug-in, from within the *Pipeline Layout Tool*, or as an independent program. Once a layout is loaded, all tasks appear as rows in a large table. Individual tasks or the system as a whole may be started, stopped, or restarted (A). The status of each task is reported (B) in the table. When a user selects a task, all inputs and dependencies for the task are shown in the ancestor pane (C), and once a task has run, its outputs are shown in the result pane (D). Debugging and log information are accessible for running or finished tasks by clicking on a row.

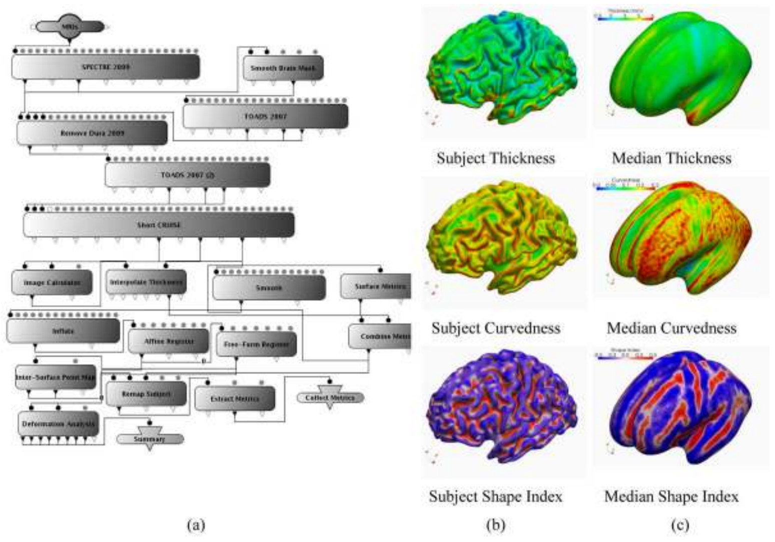


Figure 4.

All CRUISE modules were ported to use the JIST API. Rather than using shell scripts to bridge programs, the Layout Tool is used to setup dependent processing steps and validate that each program's output is passing a valid input to the next program (a). This pipeline extracts cortical surfaces from three-dimensional volumetric data in a fully automated manner. It computes local shape metrics on the surfaces which may be examined on an individual basis (column b) or registered and compared in group analyses (column c).

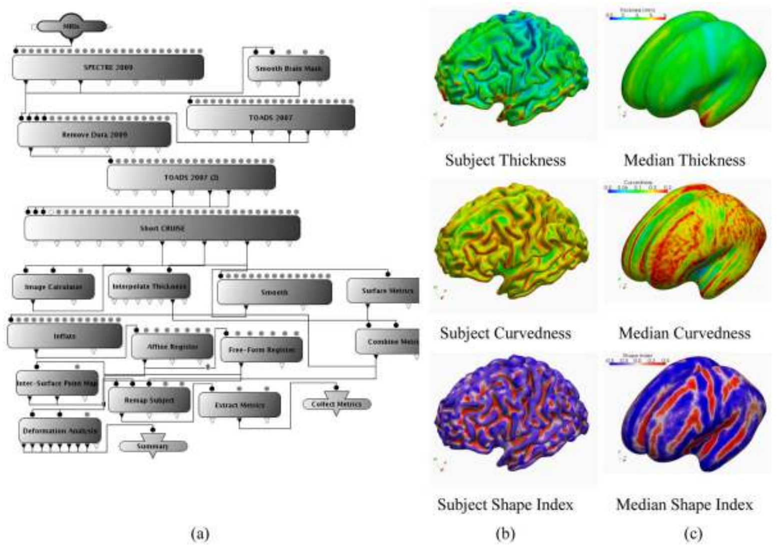


Figure 5.
The JIST port of CATNAP combines all functionality available within the original implementation as well as superior file format support and ease of integration with fiber tracking modules (a). Different methods for data modeling, motion registration, or fiber tracking can be simply substituted and explored with the pipeline environment. A visualization of all fibers overlaid with the labeled, right cortical hemisphere is shown in (b) as a typical result which is achievable with this framework. The visualization was accomplished by generating colored vector graphics surfaces and streamlines (in "vtk" format) with JIST and rendering the results in ParaView.

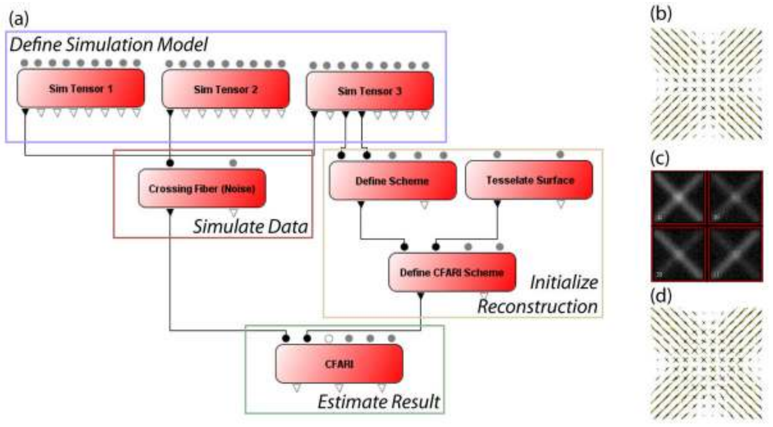


Figure 6.
JIST provided an ideal simulation framework for design of the CFARI fiber crossing analysis method. Each step in the simulation (a) could be constructed with modular units for simple reuse. In the simulation process, a noise-free biophysical model was constructed (b and top row in a). Noisy data were then simulated based on this model (c and center left in a). A discrete basis set was constructed for simulation (center right in a). Finally, the simulated data were projected onto the basis set to estimate a fiber crossing model (d and lower row in a).

Table 1

Comparison of rapid prototyping and pipelining environments.

	AVS	**SCIRun**	**OpenDX**	**LONI**	**FisWidgets**	**NA-MIC Kit**	**XIP**	**JIST**
Objective	Visualization	Visualization	Visualization	Analysis	Analysis	Analysis & Visualization	Analysis & Visualization	Analysis
GUI	Vis Prog	Vis Prog	Vis Prog	Vis Prog	Desktop	Desktop	Vis Prog	Vis Prog
Extension Mechanisms	Plugins & Binaries	Plugins	Plugins	Binaries	Binaries	Plugins and Binaries	Plugins	Plugins and Binaries
Medical Image Analysis API	No	Limited	No	No	No	ITK/VTK	ITK/VTK	MIPAV
Primary API Language	C/C++, .NET	C/C++, TCL	N/A	N/A	N/A	C/C++, Python	C/C++,	Java
Distributed Computing	Yes	Yes	No	Yes	No	Yes (via batchmake)	Yes	Yes
License	Commercial	Open Source (MIT)	Open Source (OpenDX)	Research Only (LONI)	Open Source (GPL)	Open Source (Slicer)	Open Source (caBIG)	Open Source (LGPL)