



PROVENANCE FOR VISUALIZATIONS

REPRODUCIBILITY AND BEYOND

By Claudio T. Silva, Juliana Freire, and Steven P. Callahan

The demand for the construction of complex visualizations is growing in many disciplines of science, as users are faced with ever increasing volumes of data to analyze. The authors present VisTrails, an open source provenance-management system that provides infrastructure for data exploration and visualization.

Computing has been an enormous accelerator for science, leading to an information explosion in many different fields. Future scientific advances depend on our ability to comprehend the vast amounts of data currently being produced and acquired. To analyze and understand this data, though, we must assemble complex computational processes and generate insightful visualizations, which often require combining loosely coupled resources, specialized libraries, and grid and Web services. Such processes could generate yet more data, adding to the information overflow scientists currently deal with.

Today, the scientific community uses ad hoc approaches to data exploration, but such approaches have serious limitations. In particular, scientists and engineers must expend substantial effort managing data (such as scripts that encode computational tasks, raw data, data products, images, and notes) and recording *provenance* information (that is, all the information necessary to reproduce a certain piece of data) so that they can answer basic questions: Who created a data product and when? When was it modified, and who modified it? What process was used to create the data product? Were two data products derived from the same raw data? This process is not only time-consuming, but also error-prone.

Without provenance, it's difficult (and sometimes impossible) to reproduce and share results, solve problems collaboratively, validate results with different input data, and understand the process used to solve a particular problem. In addition, data products' longevity becomes limited—without precise and sufficient information about how the data product was generated, its value diminishes significantly.

The lack of adequate provenance support in visualization systems motivated us to build VisTrails, an open source provenance-management system that provides infrastructure for data exploration and visualization through workflows. VisTrails transparently records detailed provenance of exploratory computational tasks and leverages this information beyond just the ability to reproduce and share results. In particular, it uses this information to simplify the process of exploring data through visualization.

Visualization Systems

Visualization systems such as MayaVi (<http://mayavi.sourceforge.net>) and ParaView (www.paraview.org)—which are built on top of Kitware's Visualization Toolkit (VTK)¹—as well as SCIRun (<http://software.sci.utah.edu/scirun.html>) enable users to interactively create and manipulate complex visualizations. Such systems are based

on the notion of data flows,² and they provide visual interfaces for producing visualizations by assembling *pipelines* out of modules (or functions) connected in a network. SCIRun supports an interface that lets users directly edit data flows. MayaVi and ParaView have a different interaction paradigm that implicitly builds data flows as the user makes “task-oriented” choices (such as selecting an isosurface value).

Although these systems let users create complex visualizations, they lack the ability to support data exploration at a large scale. Notably, they don't adequately support collaborative creation and exploration of multiple visualizations. Because these systems don't distinguish between the definition of a data flow and its instances, to execute a given data flow with different parameters (for example, different input files), users must manually set these parameters through a GUI. Clearly, this process doesn't scale to more than a few visualizations at a time. Additionally, modifications to parameters or to a data flow's definition are destructive—the systems don't maintain any change history. This requires the user to first construct the visualization and then remember the input data sets, parameter values, and the exact dataflow configuration that led to a particular image.

REPRODUCIBILITY AND SHARING DATA AND PROCESSES FOR THE VISUALIZATION CORNER

By Claudio Silva and Joel E. Tohline

Greetings! We're the new co-editors for the Visualization Corner. Claudio is a computer science professor at the University of Utah and faculty member of the Scientific Computing and Imaging Institute, where he does research primarily in visualization, graphics, and applied geometry. Joel is a professor of physics and astronomy at Louisiana State University and a faculty member in LSU's Center for Computation and Technology, with a research focus on complex fluid flows in astrophysical systems. We both have extensive experience in high-performance computing. In partnership with our readers and colleagues, we hope to bring you relevant and effective information about visualization techniques that can directly affect the way our readers do science. We would like to use new Web technologies (Wikis, blogs, and so on) to encourage the community to

more actively participate in the way we do things.

This first column discusses the benefits of provenance and makes a case that better provenance mechanisms are needed for visualization. In upcoming installments, we'll attempt to inform the scientific community at large about the benefits and technologies related to provenance. In particular, we want to promote the idea of reproducible visualizations. We encourage authors of articles published here to provide metadata for visualizations in their articles that let readers reproduce images as well as generate related ones (for example, using different data). Ultimately, our hope is that this trend will spread to the point that published articles will contain not only textual descriptions of the techniques, but links to data, code, and the complete overall process used to generate the scientific results.

As a mechanism to capture and share provenance metadata, authors can use VisTrails to produce specifications of the figures and plots presented in their articles. We'll archive this information at www.vistrails.org/index.php/CiSE. The data and processes associated with this column are already available on the Web site, so you can reproduce them, right now, from your desktop!

Finally, before constructing a visualization, users must often acquire, generate, or transform a given data set—for example, to calibrate a simulation, they must obtain data from sensors, generate data from a simulation, and finally construct and compare the visualizations for both data sets. Most visualization systems, however, don't give users adequate support for creating complex pipelines that support multiple libraries and services.

VisTrails: Provenance for Visualization

The VisTrails system (www.vistrails.org) we developed at the University of Utah is a new visualization system that provides a comprehensive provenance-management infrastructure and can be easily combined with existing visualization libraries. Unlike previous systems, VisTrails uses an action-based provenance model that uniformly captures changes to both parameter values and pipeline definitions by unobtrusively tracking all changes that users make to pipelines in an exploration task. We refer to

this detailed provenance of the pipeline evolution as a visualization trail, or *vistrail*.

The stored provenance ensures that users will be able to reproduce the visualizations and lets them easily navigate through the space of pipelines created for a given exploration task. The VisTrails interface lets users query, interact with, and understand the visualization process's history. In particular, they can return to previous versions of a pipeline and change the specification or parameters to generate a new visualization without losing previous changes.

Another important feature of the action-based provenance model is that it enables a series of operations that greatly simplify the exploration process and could reduce the time to insight. In particular, the model allows the flexible reuse of pipelines and provides a scalable mechanism for creating and comparing numerous visualizations as well as their corresponding pipelines. Although we originally built VisTrails to support exploratory visualization tasks, its

extensible infrastructure lets users integrate a wide range of libraries. This makes the system suitable for other exploratory tasks, including data mining and integration.

Creating an Interactive Visualization with VisTrails

To illustrate the issues involved in creating visualizations and how provenance can aid in this process, we present the following scenario, common in medical data visualization.

Starting from a volumetric computed tomography (CT) data set, we generate different visualizations by exploring the data through volume rendering, isosurfacing (extracting a contour), and slicing. Note that with proper modifications, this example also works for visualizing other types of data (for example, tetrahedral meshes).

Dataflow-Processing Networks and Visual Programming

A useful paradigm for building visualization applications is the *dataflow model*. A data flow is a directed graph in which nodes represent computations,

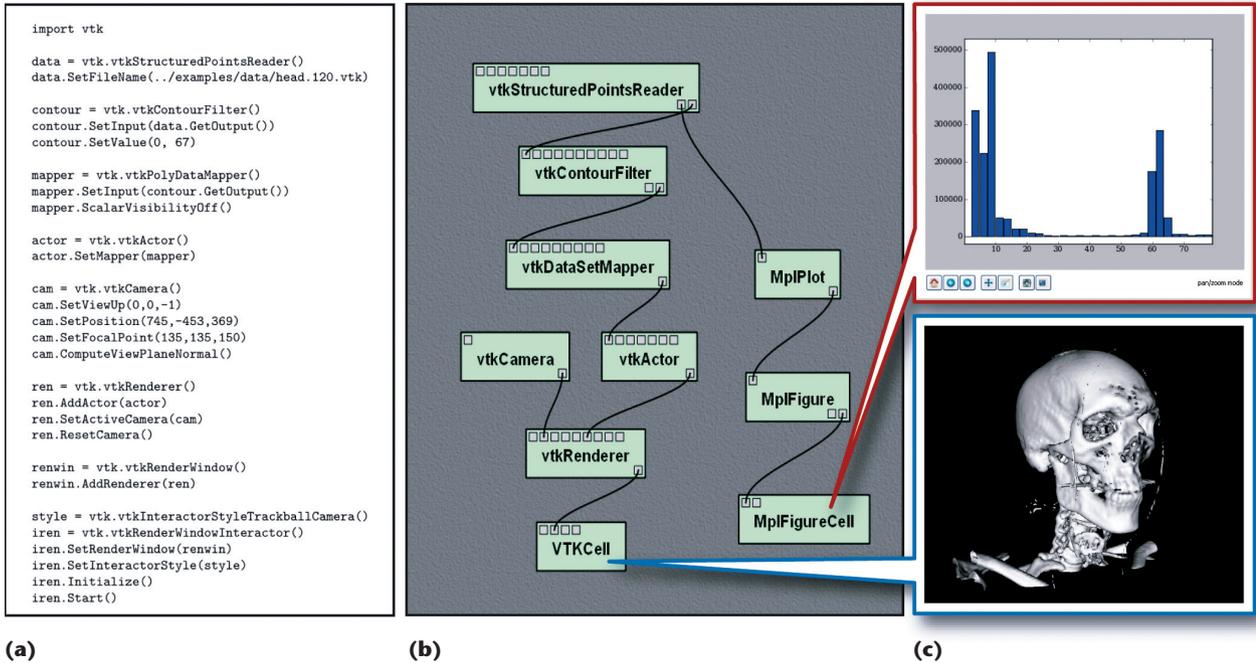


Figure 1. Dataflow programming for visualization. (a) We commonly use a script to describe a pipeline from existing libraries such as the Visualization Toolkit (VTK). (b) Visual programming interfaces, such as the one VisTrails provides, facilitate the creation and maintenance of these dataflow pipelines. The green rectangles represent modules, and the black lines represent connections. (c) The end result of the script or VisTrails pipeline is a set of interactive visualizations.

and edges represent data streams: each node or module corresponds to a procedure that’s applied on the input data and generates some output data as a result. The flow of data in the graph determines the order in which a dataflow system executes the processing nodes. In visualization, we commonly refer to a dataflow network as a visualization pipeline. (For this article, we use the terms workflow, data flow, and pipeline interchangeably.) Figure 1b shows an example of the data flow used to derive the images shown in Figure 1c. The green rectangles represent modules, and the black lines represent connections. Most of the modules in Figure 1 are from VTK, and labels on each module indicate the corresponding VTK class. In this figure, we naturally think of data flowing from top to bottom, eventually being rendered and presented for display.

We can use different mechanisms

for creating visualization pipelines—for example, scripting in a modern dynamic language, such as Python. Consider Figure 1a, which defines the workflow via a script written in Python that uses VTK to read a volume data set from a file, extract an isosurface, map the isosurface to renderable geometry, and then finally render it in an interactive window.

Visual programming interfaces for designing data flows have become popular and several systems, such as SCIRun, have adopted them. These interfaces give users a more intuitive view of the pipeline. They also dynamically perform type checking and guide the connection between modules’ input and output ports—once the user selects a module’s output, connections are allowed only to the target module’s appropriate input. VisTrails automatically pulls edges toward the correct input port. As we discuss later, another benefit of hav-

ing a high-level, structured workflow description is that we can use expressive languages in querying and updating workflows.

Comparing and Exploring Multiple Visualizations

Regardless of the specific mechanism we use to define a pipeline, the visualization process’s end goal is to gain insight from the data. To obtain such insight, users must often generate and compare multiple visualizations. Going back to our scenario, several alternatives exist for rendering our CT data. Isosurfacing is a commonly used technique. Given a function $f: \mathbf{R}^n \rightarrow \mathbf{R}$ and a value a , an isosurface consists of the set of points in a domain that map to a —that is, $S_a = \{x \in \mathbf{R}^n: f(x) = a\}$.

The range of a values determines all possible isosurfaces that the user can generate. To identify “good” a values that represent a data set’s important

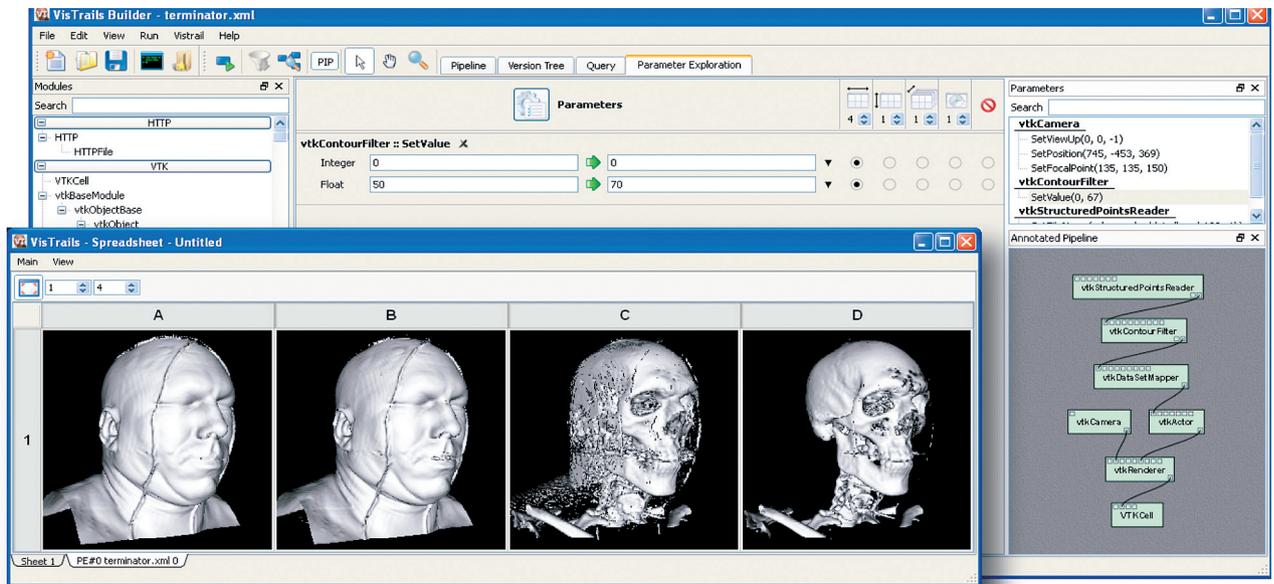


Figure 2. VisTrails' parameter exploration interface. The system computes the results efficiently by avoiding redundant computation and displays them in the spreadsheet for interactive comparative visualization.

features, we can look at the range of values taken by a , and their frequency, in the form of a histogram. Using VisTrails, we can straightforwardly extend the isosurface pipeline to also display a histogram of the data. VisTrails provides a very simple plugin functionality that you can use to add packages and libraries, including your own. For our example, we used matplotlib's 2D plotting functionality (<http://matplotlib.sourceforge.net>) to generate the histogram at the top of Figure 1c. This histogram helps in data exploration by suggesting regions of interest in the volume. The plot shows that the highest frequency features lie between the ranges [0,25] and [58,68]. To identify the features that correspond to these ranges, we must explore these regions directly through visualization.

Scalable exploration of parameter spaces. VisTrails provides an interface for parameter exploration that lets users specify a set of parameters to explore, as well as how to explore, group, and display them. As a simple 1D example, Figure 2 shows an exploration of the

isosurface value as four steps between 50 and 70, displayed horizontally in the VisTrails spreadsheet.

This spreadsheet lets us compare visualizations in different dimensions (row, column, sheet, and time); we can also link the spreadsheet's cells to synchronize the interactions between visualizations. Note that VisTrails leverages the dataflow specifications to identify and avoid redundant operations. By using the same cache for different cells in the spreadsheet, VisTrails lets users efficiently explore numerous related visualizations.

Comparing different visualization techniques. Volume rendering is a powerful computer graphics technique for visualizing 3D data. Whereas many visualization algorithms focus on creating a rendering of surfaces—although they might be surfaces of 3D objects—volume rendering lets us see “inside” the volume. This technique models the volume as cloud-like cells of semitransparent material. A surface rendering of the human body might show only the skin, for example, but a complete volume ren-

dering might also show the bones and internal organs.

Volume rendering and isosurfacing are complementary techniques, and they can generate very similar imagery depending on parameters. In fact, distinguishing between them can be difficult. The VisTrails system lets us compare workflows using a visual difference interface. To demonstrate this capability, we compute the difference between the original isosurface-generation pipeline and the new volume-rendering pipeline. Figure 3 shows the visual difference of the workflows that we can inspect, along with their resulting visualizations. In Figure 3a, we use volume rendering to create the image, in which we can see the skin on top of the bone structure; Figure 3b shows only the bone structure rendered with our standard isosurface technique. This ability to (efficiently) compare workflows and visualizations is one of the benefits of the VisTrails action-based provenance model and becomes increasingly important as a workflow becomes more complex and is shared among collaborators.

With other workflow systems, these

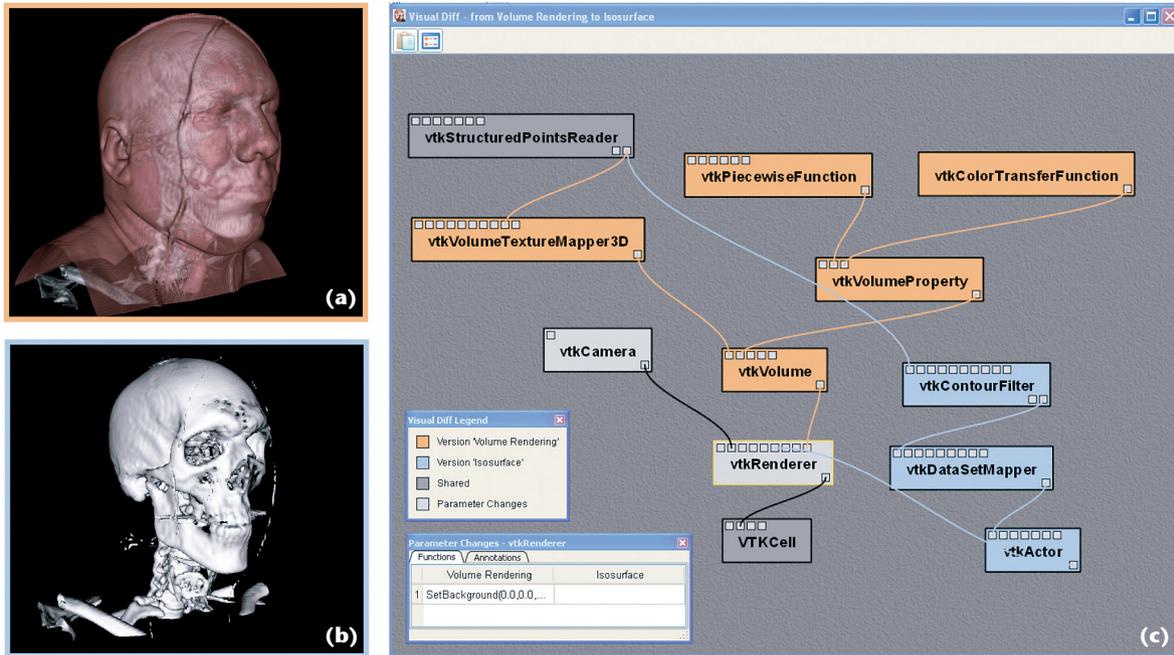


Figure 3. A visual difference between different pipelines in VisTrails. We show the difference between pipelines that generated (a) volume rendering and (b) isosurface visualizations. (c) The interface distinguishes shared modules in dark gray, the modules unique to isosurfacing in blue, those unique to direct volume rendering in orange, and those with parameter changes in light gray.

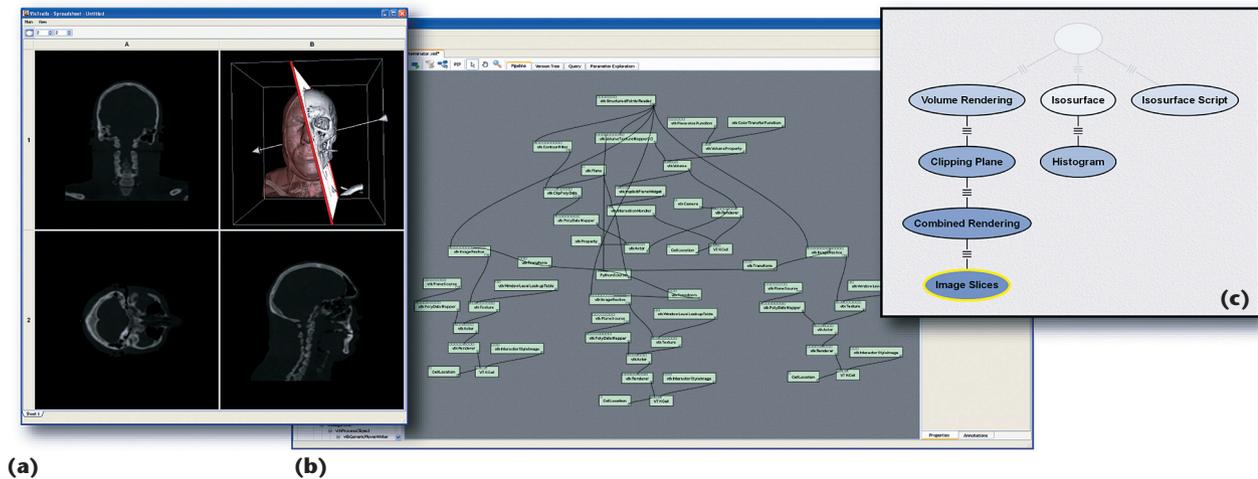


Figure 4. Multiple rendering techniques. (a) VisTrails renders visualizations by combining volume rendering and isosurfacing and updates them with user interactions. (b) The corresponding pipeline represents the data flow for creating interactive visualizations. (c) VisTrails provides a fully browsable history of the exploration process that led to this final set of visualizations.

comparisons are challenging because they require module-by-module (visual programming) or line-by-line

(scripting) comparisons. Although the former can be computationally intractable (the related decision problem of

subgraph isomorphism is NP-complete), the latter could lead to results that are hard to interpret.

THE VISTRAILS SYSTEM

In this article, we focus on using VisTrails as a tool for exploratory visualization. Additional features might be relevant for *CISE* readers:

- *Flexible provenance architecture.* VisTrails transparently tracks changes made to workflows by maintaining a detailed record of all the steps followed in the exploration. The system can optionally track runtime information about workflow execution (such as who executed a module, on which machine, and how much time elapsed). VisTrails also provides a flexible annotation framework through which users can specify application-specific provenance information.
- *Querying and reusing history.* Provenance information is stored in a structured way. Users have the choice of using a relational database (such as MySQL or IBM DB2) or XML files in the file system. The system provides flexible and intuitive query interfaces through which users can explore and reuse provenance information. Users can formulate simple keyword-based and selection queries (find a visualization that a given user created, for example) as well as structured queries (find visualizations that apply simplification before an isosurface computation for irregular grid data sets).
- *Support for collaborative exploration.* Users can configure the system with a database back end that can act as a shared repository. This back end also provides a synchronization facility that lets users collaborate asynchronously and in a disconnected fashion—they can check changes in and out, akin to a version-control system (such as subversion (SVN), <http://subversion.tigris.org>).

- *Extensibility.* VisTrails provides a very simple plug-in functionality that can help dynamically add packages and libraries. Neither changes to the user interface nor system recompilation are necessary. Because VisTrails is written in Python, integrating Python-wrapped libraries is straightforward.
- *Scalable derivation of data products and parameter exploration.* VisTrails supports a series of operations for simultaneously generating multiple data products, including an interface that lets users specify sets of values for different parameters in a workflow. Users can display the results of a parameter exploration side by side in the VisTrails spreadsheet for easy comparison.
- *Task creation by analogy.* VisTrails supports analogies as first-class operations to guide semiautomated changes to multiple workflows, without requiring users to directly manipulate or edit the workflow specifications.

Please visit www.vistrails.org to access the VisTrails community Web site. You'll find information including instructions for obtaining the software, online documentation, video tutorials, and pointers to papers and presentations.

VisTrails is written in Python and uses the multiplatform Qt library for its user interface. The system is open source, released under the GPL 2.0 license. The pre-compiled versions for Windows, Mac OS X, and Linux come with an installer and several packages, including VTK, matplotlib, and Image Magick. Additional packages, including ones users have written, are also available, but you can easily add new packages using the VisTrails plug-in infrastructure. Detailed instructions are available at our Web site.

Interacting with visualizations. The images we generated so far correspond to simple, static workflows. To perform a more dynamic comparison between volume rendering and isosurfacing, we add a feedback loop into the workflow to let users adjust the visualization interactively. We thus build a new workflow that uses the isosurfacing and volume rendering algorithms simultaneously. We add a clipping plane into the volume visualization to assign the volume regions used for each algorithm. In addition, we use a point on the plane to define axis-aligned slices of the volume that we display in distinct spreadsheet cells. The pipeline in-

teractively updates these slices along with the plane during user interactions. Figure 4 shows the resulting visualizations along with the complex dynamic workflow required to produce them.

Provenance and Exploratory Visualization

The combination of multiple visualization algorithms, different libraries, and the interactions between them considerably complicates the workflow specification. In addition, creating a set of visualizations from data is not always a linear process and often involves several iterations as a user formulates and tests

hypotheses. Whereas for simple experiments, manual approaches to provenance management might be feasible, complex computational tasks involving large volumes of data or multiple researchers require automated approaches. As these tasks' complexity and scale increases, data organization becomes a major component of the process.

VisTrails manages the data manipulated and metadata created in the course of an exploratory task.³ As a user (or group of users) generates a series of visualizations, VisTrails transparently tracks all the steps this exploration followed—that is, the modules and connections added and

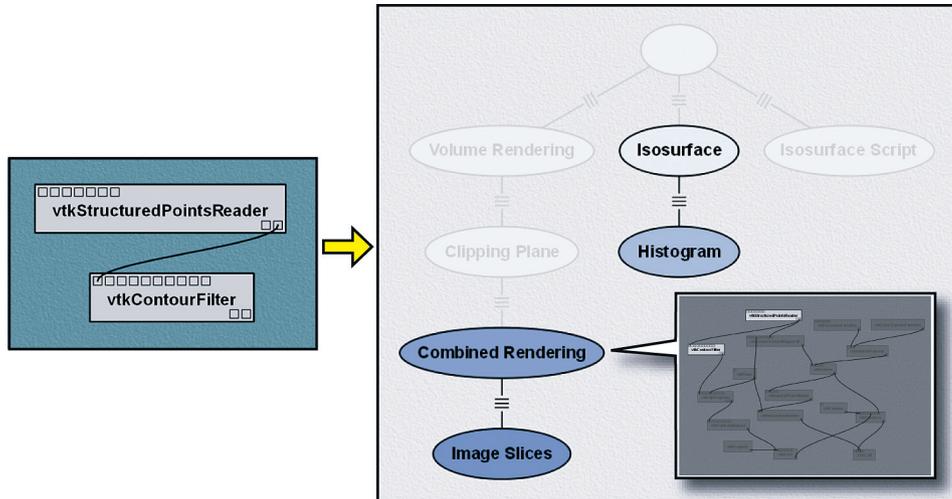


Figure 5. The VisTrails query-by-example interface. (a) Users can define a set of modules and parameters in the visual programming interface to create a query template. (b) The query results are shown in the history tree, which users can browse for specific instances of the match (inset).

Table 1. Query examples.	
Query	Result
<i>volume</i>	Highlights all nodes in the history tree in which the string “volume” appears (for example, in a module name, parameter name, annotation)
<i>user:juliana</i>	Highlights all nodes in the history tree created by the user “juliana”
<i>before: March 30, 2007</i>	Highlights all nodes in the history tree created before “March 30, 2007”

- seamlessly navigate over the history tree and return to previous pipeline versions after reaching a dead end;
- undo bad changes;
- reuse pipelines and pipeline fragments from previous versions;
- compare different pipelines and their results; and
- be reminded, intuitively, of the actions that led to a particular result.

deleted, parameter value changes, and so on. Figure 4c shows a history tree of the different pipelines created in the course of our running example. The nodes in this tree correspond to pipelines; an edge between two pipelines corresponds to changes performed on the parent pipeline to obtain its child. For readability, by default, only the nodes in the tree that the user tags as important are displayed.

By tracking all the changes made to a workflow ensemble, VisTrails properly captures each step, leaving a complete trail of the work. Having access to the different pipelines’ specifications lets others reproduce and share the results of each step in the exploratory process. To demonstrate this, we made the vistrail asso-

ciated with this example available for download with the VisTrails system (see “The VisTrails System” sidebar). You can recreate each figure shown in this article by executing the different nodes in the history tree. Note that by using the action-based provenance model, we obtain a very concise representation of the history, which uses substantially less space than the alternative of explicitly storing multiple *versions* of a pipeline.³

The exploration trail VisTrails captures also supports various activities that are crucial for performing reflective reasoning and obtaining insights, such as following chains of reasoning backward and forward and comparing different results.⁴ The tree-based view lets users

Thus, users can efficiently explore several related visualizations.

The issue of reproducibility for visualization has been considered before,⁵ but we should note that whereas some visualization and workflow systems provide support for provenance tracking, their focus has been on data provenance—that is, information about how the system derived a given data product, including the parameter values used⁶—and on interaction provenance (such as capturing a visualization’s viewing manipulations).⁷ VisTrails is the first system to capture information about how workflows evolve over time.

For instance, to generate the composite visualization in our final example, we extended our pipeline labeled Volume Rendering to include

modules from the pipeline labeled Isosurfacing. (Having two pipelines lets us further explore the visualization—by trying different isosurface values, for example [see Figure 2]). In addition, we can compare the pipelines by dragging one node on top of the other (see Figure 3). These computed differences are useful for understanding the visualization process, and the user can also reuse them. In this case, we applied the modules unique to “Isosurfacing” to “Volume Rendering” to create a new pipeline called “Combined Rendering,” that uses a cutting plane to define regions for the rendering methods. VisTrails can automatically apply pipeline differences (like a patch) to derive new pipelines in a process we call *visualization creation by analogy*.⁸

Another benefit to having a high-level specification of the visualization process is that users can query the pipelines and their execution instances. Scientists can query a vis-trail to find anomalies in previously generated visualizations and locate data products and visualizations based on operations applied in the visualization process. VisTrails supports simple, keyword-based queries as well as structured queries. In addition to providing information about the results (for example, workflow identifiers and attributes), VisTrails can visually display query results by highlighting the workflows and modules that satisfy the query. Table 1 shows an example.

Users might also define queries by example.⁸ As Figure 5 illustrates, users can construct (or copy and paste) a pipeline fragment into the VisTrails query tab to identify in the history tree all nodes that contain that fragment. They can then browse through the highlighted nodes and click on one

to display the workflow and highlight the modules that match the query. Users can then click on the individual modules to view the execution log records associated with them.

The VisTrails project has focused on creating an infrastructure to manage the provenance data of exploratory tasks. With this infrastructure in place, our research focus is now on what we can do with all the provenance accumulation. By mining this information, we hope to learn useful patterns that can guide users in assembling and refining complex computational tasks. 

Acknowledgments

This article summarizes work being done in the VisTrails project. It's only possible through the work of all our team members: Erik Anderson, Jason Callahan, David Koop, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. The data used in this article is available courtesy of the National Library of Medicine's Visible Human Project. The US National Science Foundation partially supported this work under grants IIS-0513692, CCF-0401498, EIA-0323604, CNS-0541560, OCE-0424602, and OISE-0405402. The US Department of Energy, an IBM Faculty Award, and a University of Utah Seed Grant also partially supported this work.

References

1. W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*, Kitware, 2003.
2. E.A. Lee and T.M. Parks, “Dataflow Process Networks,” *Proc. IEEE*, vol. 83, no. 5, 1995, pp. 773–801.
3. S. Callahan et al., “Managing the Evolution of Dataflows with VisTrails (extended abstract),” *Proc. IEEE Workshop on Workflow and*

Data Flow for Scientific Applications (SciFlow), IEEE CS Press, 2006.

4. D.A. Norman, *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*, Addison-Wesley, 1994.
5. G. Kindlmann, “Lack of Reproducibility Hinders Visualization Science,” *IEEE Visualization Compendium*, IEEE CS Press, 2006, p. 69.
6. T. Jankun-Kelly, K.-L. Ma, and M. Gertz, “A Model and Framework for Visualization Exploration,” *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 2, 2007, pp. 357–369.
7. D.P. Groth and K. Streefkerk, “Provenance and Annotation for Visual Exploration Systems,” *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 6, 2006, pp. 1500–1510.
8. C. Scheidegger et al., “Querying and Creating Visualizations by Analogy,” to be published in *IEEE Trans. Visualization and Computer Graphics*, 2007.

Claudio T. Silva is an associate professor at the University of Utah. His research interests include visualization, geometry processing, graphics, and high-performance computing. Silva has a PhD in computer science from SUNY at Stony Brook. He is a member of the IEEE, the ACM, Eurographics, and Sociedade Brasileira de Matematica. Contact him at csilva@cs.utah.edu.

Juliana Freire is an assistant professor at the University of Utah. Her research interests include scientific data management, Web information systems, and information integration. Freire has a PhD in computer science from SUNY at Stony Brook. She is a member of the ACM and the IEEE. Contact her at juliana@cs.utah.edu.

Steven P. Callahan is a research assistant and PhD candidate at the University of Utah. His research interests include scientific visualization, visualization systems, and computer graphics. Callahan has an MS in computational engineering and science from the University of Utah. Contact him at stevec@sci.utah.edu.