Using Provenance to Streamline Data Exploration through Visualization

Steven P. Callahan Juliana Freire Emanuele Santos
Carlos E. Scheidegger Cláudio T. Silva Huy T. Vo
SCI Institute and School of Computing – University of Utah

ABSTRACT

Scientists are faced with increasingly larger volumes of data to analyze. To analyze and validate various hypotheses, they need to create insightful visual representations of both observed data and simulated processes. Often, insight comes from comparing multiple visualizations. But data exploration through visualization requires scientists to assemble complex workflows—pipelines consisting of sequences of operations that transform the data into appropriate visual representations—and today, this process contains many error-prone and time-consuming tasks.

We show how a new action-based model for capturing and maintaining *detailed provenance of the visualization process* can be used to streamline the data exploration process and reduce the time to insight. This model enables the flexible re-use of workflows, a scalable mechanism for creating a large number of visualizations, and collaboration in a distributed setting. A novel feature of this model is that it uniformly captures provenance information for both visualization data products and workflows used to generate these products. By also tracking the evolution of workflows, it not only ensures reproducibility, but also allows scientists to easily navigate through the space of workflows and parameter settings used in a given exploration task. We describe the implementation of this data exploration infrastructure in the VisTrails system, and present two case studies which show how it greatly simplifies the scientific discovery process.

1. INTRODUCTION

Computing is an enormous accelerator to science and it has led to an information explosion in many different fields. Future advances in science depend on the ability to comprehend these vast amounts of data being produced and acquired. Visualization is a key enabling technology in this endeavor [15]—it helps people explore and explain data through software systems that provide a static or interactive visual representation. A basic premise of visualization is that visual information can be processed at a much higher rate than raw numbers and text.

Despite the promise that visualization can serve as an effective *enabler* of advances in other disciplines, the application of visualization technology is non-trivial. The design of effective visualizations is a complex process that requires deep understanding of existing techniques, and how they relate to human cognition. Although there have been enormous advances in the area, the use of advanced visualization techniques is still limited. A key barrier to the effective use of visualization is the lack of appropriate data management techniques needed for scalable data exploration and hypothesis testing. In this paper, we propose a novel infrastructure whose goal is to simplify and streamline the process of data exploration through visualization.

Data Exploration Through Visualization. To successfully analyze and validate various hypotheses, it is necessary to pose several queries, correlate disparate data, and create insightful visualizations of both the simulated processes and observed phenomena. However, data exploration through visualization requires scientists to go through several steps. As illustrated in Figure 1, they need to assemble and execute complex workflows that consist of data set selection, specification of series of operations that need to be applied to the data, and the creation of appropriate visual representations, before they can finally view and analyze the results. Often, insight comes from comparing the results of multiple visualizations created during the exploration process. For example, by applying a given visualization process to multiple datasets generated in different simulations; by varying the values of certain visualization parameters; or by applying different variations of a given process (*e.g.,* which use different visualization algorithms) to a dataset. Unfortunately, today this exploratory process contains many manual, error-prone, and time-consuming tasks. As a result, scientists spend much of their time managing the data rather than using their time effectively for scientific investigation and discovery.

Because this process is complex, and requires deep understanding of both visualization techniques and a particular scientific domain, it requires close collaboration among domain scientists and visualization experts [15]. Thus, adequate support for collaboration is key to fully explore the benefits of visualization.

Visualization Systems: The State of the Art. Visualization systems such as Paraview [16] and SCIRun [24] allow the interactive creation and manipulation of complex visualizations. These systems are based on the notion of dataflows [19], and they provide visual interfaces to produce visualizations by assembling *pipelines*[1] out of modules connected in a network. Although these systems allow the creation of complex visualizations, they have important limitations which hamper their ability to support the data exploration process at a large scale. In particular, they lack scalable mechanisms for the exploration of parameter spaces. In addition,

[1]In the remainder of this paper, we use the terms pipeline and dataflow interchangeably.

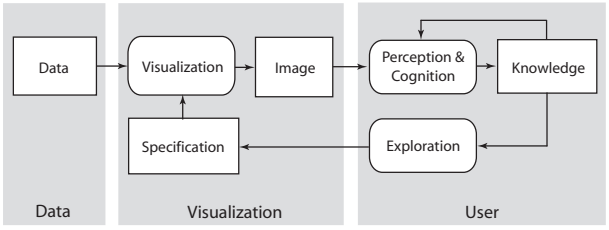


Figure 1: The visualization discovery process. Scientists first select the data to visualize, and then specify the algorithms and visualization techniques to visualize the data. This specification is adjusted in an interactive process, as the scientist generates, explores and evaluate hypotheses about the information under study. This figure is adapted from [15,36].

whereas they manage and simplify the process of creating visualizations, they lack infrastructure to manage the data involved in the process—input data, metadata, and derived data products. Consequently, *they do not provide adequate support for the creation and exploration of a large number of visualizations*—a key requirement in the exploration of scientific data. Last, but not least, they provide no support for collaborative creation and manipulation of visualizations.

Many of these limitations stem from the fact that *these systems do not distinguish between the definition of a dataflow and its instances*. In order to execute a given dataflow with different parameters (*e.g.,* different input files), users need to manually set these parameters through a GUI. Clearly, this process does not scale to more than a few visualizations. Additionally, modifications to parameters or to the definition of a dataflow are destructive—*no change history is maintained*. This places the burden on the scientist to first construct the visualization and then to remember the input data sets, parameter values, and the exact dataflow configuration that led to a particular image. Finally, they *do not exploit optimization opportunities during pipeline execution*. For example, SCIRun not only materializes all intermediate results, but it may also repeatedly recompute the same results over and over again if so defined in the visualization pipeline. As a result, the creation, maintenance, and exploration of visualization data products are major bottlenecks in the scientific process, limiting the scientists' ability to fully exploit their data.

The VisTrails System. In VisTrails [3, 6, 7], we address the problem of visualization from a data management perspective: VisTrails *manages* both the visualization process and the metadata associated with visualization products. It provides infrastructure that enables interactive multiple-view visualizations by simplifying the creation and maintenance of visualization pipelines, and by optimizing their execution.

In a nutshell, VisTrails consists of a scientific workflow middleware which can be combined with existing visualization systems and libraries (*e.g.,* SCIRun [24] and Kitware's VTK [17,28]). A key component of VisTrails is a parameterized dataflow specification—a formal specification of a pipeline. Unlike existing dataflow-based visualization systems, in VisTrails there is a clear separation between the specification of a pipeline and its execution instances. This separation enables powerful scripting capabilities and provides a scalable mechanism for generating a large number of visualizations. A pipeline definition can be used as a template, and instantiated with different sets of parameters to generate several visualizations in a scalable fashion. In addition, by representing the pipeline specification in a structured way (using XML [37]), the system allows the visualization provenance to be queried and mined.

VisTrails also leverages the dataflow specification to identify and avoid redundant operations in a transparent fashion. This optimization is especially useful while exploring multiple visualizations. When variations of the same pipeline need to be executed, substantial speedups can be obtained by caching the results of overlapping subsequences of the pipelines. The high-level architecture of VisTrails, its caching mechanism, and its support for multi-view visualizations are described in [3].

Action-Based Provenance and Data Exploration. In this paper we describe a new action-based provenance model we designed for VisTrails. Unlike visualization systems [16, 24] and scientific workflow systems [1, 23, 33, 34], VisTrails *captures detailed provenance of the exploratory process*. It *unobtrusively* records all user interactions with the system. And because the unit of provenance is a user action, this model uniformly captures changes to parameter values and to workflow definitions. The stored provenance ensures reproducibility of the visualizations, and it also allows scientists to easily navigate through the space of dataflows created for a given exploration task. In particular, giving them the ability to return to previous versions of a dataflow and/or different parameter settings and comparatively visualize their results. We also describe how the action-based model can be used to streamline the data exploration process and reduce the time to insight by enabling the flexible reuse of workflows, a scalable mechanism for creating a large number of visualizations, and collaboration in a distributed setting.

Outline and Contributions. This paper provides the first detailed description of the VisTrails provenance and data exploration infrastructure. In Section 2, we use real application scenarios to illustrate the complexities involved in exploring data through visualization and how they greatly hinder the scientific discovery process. A brief overview of VisTrails is given in Section 3. In Section 4 we present our action-based provenance mechanism, and in the two sections following that, we show how useful data exploration operations can be implemented within this framework. In Section 5 we describe general mechanisms for enabling pipeline re-use, and scalable and simplified generation of visualizations. A scheme that enables VisTrails to be used as a collaborative platform for data exploration is presented in Section 6. We discuss in Section 7 how the VisTrails data exploration infrastructure positively impacts and simplifies the visualization tasks in two application domains: environmental sciences and radiation oncology treatment planning.

2. MOTIVATING EXAMPLES

Below, we describe two applications that motivated us to develop VisTrails. The examples illustrate some of the problems faced by our collaborators in exploring data through visualization. Although the scenarios described are in the areas of Environmental Sciences and Medical Diagnosis and Treatment Planning, these problems are common in many other domains of science.

2.1 Using Visualization to Understand the Columbia River

Understanding ecosystems by modeling environmental processes is an important area of research that brings many benefits to science and society [2]. Paradigms for modeling and visualization of complex ecosystems are changing quickly, creating enormous opportunities for those studying them. Environmental Observation and Forecasting Systems (EOFS) integrate real-time sensor networks, data management systems, and advanced numerical models to pro-

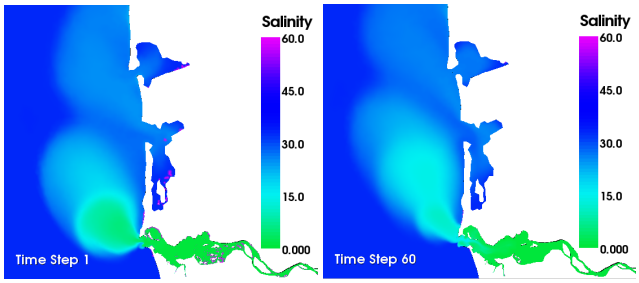


Figure 2: An example of a visualization of forecasts in the Lower Columbia River for different time steps.

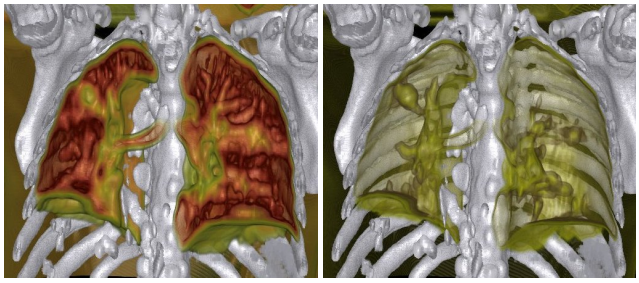


Figure 3: An example of visualization of radiation oncology in a breathing cycle of a lung. The visualizations focus on normal tissue (left) and pathological tissue (right).

vide objective insights about the environment in a timely manner. However, due to the volume of measured and forecast data, EOFS modelers are given the overwhelming task of managing a large number of visualizations.

As an example, the CORIE[2] project was established to study the spatial and temporal variability of the Lower Columbia River. Figure 2 shows an example of the results of multiple visualization techniques applied to a time steps of a forecast generated by the CORIE project.

Under the direction of Professor António Baptista, the CORIE project involves many staff members from diverse backgrounds including oceanography, database management, and computational science. The project produces and publishes thousands visualizations on a daily basis. Currently, the process of generating a visualization product is performed by a member of the staff by running a series of custom scripts and VTK [28] pipelines generated for previous forecasts. For example, to produce a composite visualization for a presentation, multiple staff members are needed to generate new scripts or modifying existing ones until suitable visualizations are found. This process is often repeated for similar and complementary simulation runs. The resulting visualizations are then composited into a single image using copy and paste functionality within PowerPoint. Usually, the figure caption and legends are the only metadata available for the composite vizualization—making it hard, and sometimes impossible, to reproduce the visualization.

The process followed by Baptista's staff is both time consuming and error prone. The process of creating and maintaining these scripts is done completely manually since there is no infrastructure for managing the scripts and associated data. Often, finding and running the scripts are tasks that can only be performed by their creators. Even for their creators, managing the scripts and data in an ad-hoc manner can be difficult because the data provenance and relationships among different scripts are not captured in a persistent way.

2.2 Using Visualization for Radiation Oncology Treatment Planning

Visualization has been used by physicians for many years to provide diagnosis as well as treatment planning in patients. The advent of medical imaging devices such as Magnetic Resonance Imaging (MRI) scanners have enabled doctors to distinguish pathologic tissue (such as a tumor) from normal tissue. Typically, a clinician navigates through a series of 2D slices to find problematic areas. Many of the state-of-the-art visualization techniques that have been developed to visualize and explore the data in 3D have not been approved for clinical use. However, some research radiologists are using the advanced techniques to quickly explore the data and assist in the location of tumors using clinically approved methods [21, 25].

Dr. George Chen is the director of the Radiation Physics Division of the Department of Radiation Oncology at the Massachusetts General Hospital. Radiation oncologists in his division have been using advanced visualization techniques on MRI data to locate tumors within the body in preparation for radiation therapy treatments. Because the results of the MRI are different for every patient, the process of creating a suitable visualization cannot be uniformly applied to every MRI data set. Instead, an iterative routine is performed between the doctor and a visualization specialist until a suitable visualization is found. First, the data is given to the visualization specialist, an initial dataflow is created, and multiple images are generated with varying parameters (*i.e.,* focusing on specific tissue or bone and using different time steps). In almost every case, this initial set of visualizations is not acceptable to the clinicians and oncologists receiving them. After receiving feedback from the the doctors, the visualization specialist adjusts parameters and changes the dataflow to increase the quality of the visualization and presents the results to the clinicians for review. This process continues until a final dataflow is established. Due to the high sensitivity of the process, very accurate and detailed records of the data manipulation are required. This documentation process produces hundreds of saved files and many detailed pages of handwritten notes. Figure 3 shows an example of a time step of an animation showing both pathological and normal tissue in a lung breathing cycle.

Visualization tools such as SCIRun [24] and Paraview [16] provide an interface to create visualization pipelines such as those used for visualizing radiation oncology, but fail to capture the process of creating a visualization. Thus, vast amounts of data are manually managed (through file saves and handwritten notes) to record the changes that take place from one visualization to another.

3. VISTRAILS: OVERVIEW

As illustrated in the examples above, our motivation for developing VisTrails came from the realization that visualization systems lack adequate support for large-scale data exploration. VisTrails is a visualization management system that manages both the process and metadata associated with visualizations. With VisTrails, we aim to give scientists a dramatically improved and simplified process to analyze and visualize large ensembles of simulations and observed phenomena.

3.1 System Architecture

The high-level architecture of the system is shown in Figure 4.

[2]http://www.ccalmr.ogi.edu/CORIE

Users create and edit dataflows using the *Vistrail Builder* user interface. The dataflow specifications are saved in the *Vistrail Repository*. Users may also interact with saved dataflows by invoking them through the *Vistrail Server* (*e.g.,* through a Web-based interface) or by importing them into the *Visualization Spreadsheet*. Each cell in the spreadsheet represents a view that corresponds to a dataflow instance; users can modify the parameters of a dataflow as well as synchronize parameters across different cells. Dataflow execution is controlled by the *Vistrail Cache Manager*, which keeps track of invoked operations and their respective parameters. Only new combinations of operations and parameters are requested from the *Vistrail Player*, which executes the operations by invoking the appropriate functions from the *Visualization and Script APIs*. The Player also interacts with the *Optimizer* module, which analyzes and optimizes the dataflow specifications. A log of dataflow executions is kept in the *Vistrail Log*. The different components of the system are briefly described in below. A more detailed description of an earlier version of our system is given in [3].

Dataflow Specifications. A key feature that distinguishes VisTrails from previous visualization systems is that it separates the notion of a dataflow specification from its instances. A dataflow instance consists of a sequence of operations used to generate a visualization. This information serves both as a log of the steps followed to generate a visualization—a record of the visualization provenance—and as a recipe to automatically re-generate the visualization at a later time. The steps can be replayed exactly as they were first executed, and they can also be used as templates—they can be parameterized. For example, the visualization spreadsheet in Figure 5(b) illustrates a multi-view visualization of a single dataflow specification varying the time step parameter. Operations in a vistrail dataflow include visualization operations (*e.g.,* VTK calls); application-specific steps (*e.g.,* invoking a simulation script); and general file manipulation functions (*e.g.,* transferring files between servers). To handle the variability in the structure of different kinds of operations, and to easily support the addition of new operations, we defined a flexible XML schema [5] to represent the dataflows. The schema captures all information required to re-execute a given dataflow. The schema stores information about individual modules in the dataflow (*e.g.,* the function executed by the module, input and output parameters) and their connections—how outputs of a given module are connected to the input ports of another module. The XML representation for vistrail dataflows allows the reuse of standard XML tools and technologies. An important benefit of using an open, self-describing specification is the ability to share (and publish) dataflows. This allows a scientist to publish an image along with its associated dataflow so that others can easily reproduce the results.

Another benefit of using XML is that the dataflow specification can be queried using standard XML query languages such as XPath [38] and XQuery [4]. For example, an XQuery query could be posed by Professor Baptista to find a dataflow that provides a 3D visualization of the salinity at the Columbia River estuary (as in Figure 5) from a database of published dataflows. Once the dataflow is found, he could then apply the same dataflow to more current simulation results, or modify the dataflow to test an alternative hypothesis. With VisTrails, he has the ability to steer his own simulations.

Caching, Analysis and Optimization. Having a high-level specification allows the system to *analyze and optimize dataflows*. Executing a dataflow can take a long time, especially if large data sets and complex visualization operations are used. It is thus important to be able to analyze the specification and identify optimization opportunities. Possible optimizations include, for example, factoring out common subexpressions that produce the same value; removing no-ops; identifying steps that can be executed in parallel; and identifying intermediate results that should be cached to minimize execution time. Although most of these optimization techniques are widely used in other areas, they have yet to be applied in dataflow-based visualization systems.

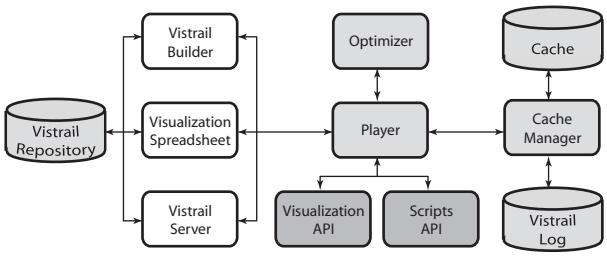


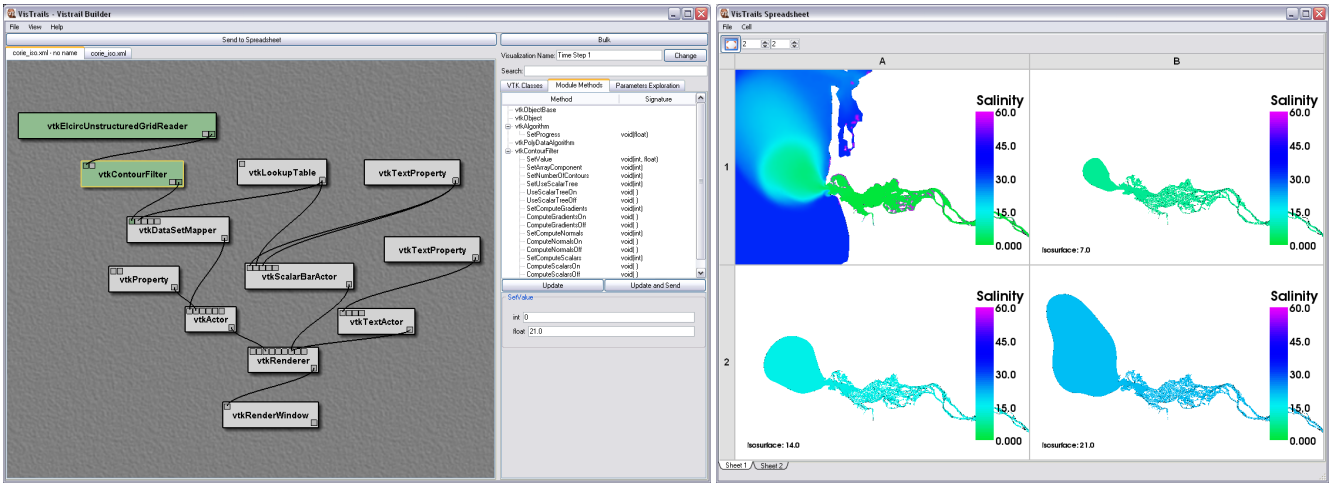
Figure 4: VisTrails Architecture.

In our current VisTrails prototype, we have implemented memoization (caching). VisTrails leverages the dataflow specification to identify and avoid redundant operations. The algorithms and implementation of the Vistrail Cache Manager (VCM) are described in [3]. Caching is especially useful while exploring multiple visualizations. When variations of the same dataflow need to be executed, substantial speedups can be obtained by caching the results of overlapping subsequences of the dataflows.

Playing a Dataflow. The Vistrail Player (VP) receives as input an XML file for a dataflow instance and executes it using the underlying Visualization or Script APIs. The semantics of each particular execution are defined by the underlying API. Currently, the VP supports VTK classes and external scripts. It is a very simple interpreter. For the more complex case of VTK, the VP directly translates the dataflow modules into VTK classes and sets their connections. Then, it sets the correct parameters for the modules according to the parameter values in the dataflow instance. Finally, the resulting network is executed by calling update methods on the sink nodes. The VP needs the ability to create and execute arbitrary VTK modules from a dataflow. This requires mapping VTK descriptions, such as class and method names, to the appropriate `module` elements in the dataflow schema. The wrapping mechanism is library-specific, and in our first version [3], we exploited VTK automatic wrapping mechanism to generate all required bindings directly from the VTK library headers. Our new implementation uses Python to further simplify the process of wrapping external libraries, and to enable easy extensions to the system.

Note that the VP is unaware of caching. To accommodate caching in the player, we use a class that behaves essentially like a proxy. To the rest of the dataflow, it looks perfectly like a filter, but instead of performing any computations, it simply looks up the result in the cache. The VCM is responsible for replacing a complex subnetwork that has been previously executed with an appropriate instance of the caching class. After the (partial) dataflow is executed, its outputs are stored in new cache entries.

Information pertinent to the execution of a particular dataflow instance is kept in the Vistrail Log (see Figure 4). There are many benefits from keeping this information, including: the *ability to debug* the application—*e.g.,* it is possible to check the results of a dataflow using simulation data against sensor data; *reduced cost of failures*—if a visualization process fails, it can be restarted from the



(a) (b)

Figure 5: The Vistrail Builder (a) and Vistrail Spreadsheet (b) showing the dataflow and visualization products of the CORIE data.

failure point. The latter is especially useful for long running processes, as it may be very expensive and time-consuming to execute the whole process from scratch. Logging *all* the information associated with all dataflows may not be feasible. VisTrails provides an interface that lets users select which and how much information should be saved.

Creating and Interacting with Vistrails. The Vistrail Builder (VB) allows users to create and edit dataflows (see Figure 5(a)). It writes (and also reads) dataflows in the same XML format as the other components of the system. It shares the familiar nodes-and-connections paradigm with dataflow systems. In order to automatically generate the visual representation of the modules, it reads the same data structure generated by the VP VTK wrapping process (see *Playing a Vistrail* above). Like the VP, *the VB requires no change to support additional types of modules*. The VB uses QT[3] and OpenGL[4] to display the dataflow of a visualization as well as the history of changes to the dataflow.

The VisTrails Visualization Spreadsheet (VS) allows users compare the results of multiple dataflows. The VS consists of a set of separate visualization windows arranged in a tabular view. This layout makes efficient use of screen space, and the row/column groupings can conceptually help the user explore the visualization parameter space [8, 9] (see Section 5.2). The cells in a spreadsheet may execute different dataflows and they may also use different parameters for the same dataflow specification (see Figure 5). To ensure efficient execution, all cells share the same cache. Note that cells in a spreadsheet can be synchronized in many different ways. For example, in Figure 8, cells are synchronized with respect to the camera viewpoint—if one cell is rotated, the same rotation is applied to the other synchronized cell. This figure also illustrates the usefulness of the spreadsheet to explore the parameter space of an application. In this case, it allows different visualizations of MRI data of a lung to be compared: different isosurface values are used in the horizontal axis, and different opacity values are used in the vertical axis.

[3]http://www.trolltech.com
[4]http://www.opengl.org

4. ACTION-BASED PROVENANCE

The first version of VisTrails only tracked provenance of visualization products [3]: for a given visualization, it stored the steps and parameters that led to the visualization. To explore data, scientists create many related visualizations that must be compared, so that they can understand complex phenomena, calibrate simulation parameters or debug applications. While working on a particular problem, scientists often create several variations of a workflow through trial and error, and these workflows may differ in both parameter values and the actual workflow specifications.

To provide full provenance of the visualization exploration process, we introduce the notion of a visualization trail, a vistrail. *A vistrail captures the evolution of a dataflow—all steps followed to construct a set of visualizations*. It represents several versions of a dataflow (which differ in their specifications), their relationships, and their instances (which differ in the parameters used in each particular execution).

VisTrails uses an action-based model to capture provenance. As the scientist makes modifications to a particular dataflow, the provenance mechanism records those changes. Instead of storing a set of related dataflows, we store the operations or actions that are applied to the dataflows. Besides being simple and compact, the action-based representation enables the construction of a user interface that shows the history of the dataflow through these changes, as can be seen in Figure 6. A tree-based view allows a scientist to return to a previous version in an intuitive way. The scientist can undo bad changes, make comparisons between datasets or parameter settings, and be reminded of the actions that led to a particular result. This, combined with a caching strategy that eliminates redundant computations, allows the scientist to efficiently explore a large number of related visualizations.

Although the issue of provenance in data management systems and in particular, for scientific workflows, has received substantial attention recently, most works focus on data provenance only, *i.e.,* maintaining information of how a given data product was generated [23, 26, 31]. VisTrails is the first system to provide a mechanism that uniformly captures provenance information for both the data and the processes that derive the data. As we discuss below, the action-based representation of provenance has several benefits: it enables the creation of intuitive mechanisms for dataflow re-use

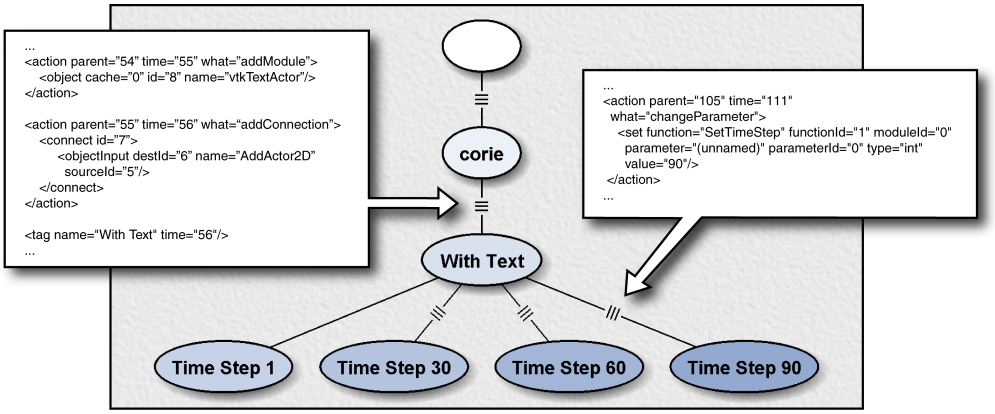


Figure 6: A snapshot of the VisTrails history management interface. Each node in the vistrail history tree represents a dataflow version. An edge between a parent and child nodes represents to a set of actions applied to the parent to obtain the dataflow for the child node.

and parameter-space exploration (Section 5), and distributed collaboration (Section 6).

Vistrail: An Evolving Dataflow. A vistrail VT is a rooted tree in which each node corresponds to a *version* of a dataflow, and each edge between nodes d_p and d_c, where d_p is the parent of d_c, corresponds to the action applied to d_p which generated d_c. This is similar to the versioning mechanism used in DARCS [27].

The vistrail in Figure 6 shows a set of changes to a dataflow that was used to generate the CORIE visualization products shown in Figure 5. In this case, a dataflow to visualize the salinity in a small section of the estuary. This dataflow (tagged "With Text") was used to create four different dataflows that represent different time steps of the data and are shown separately in the spreadsheet. Note that in this figure, only tagged nodes are displayed. Instead of displaying every version, by default we only show versions that the user tags. We represent an edge between two tagged versions in different ways. If a tagged version is not a child of another, the edge will represent a series of actions, and we draw this as an edge crossed with three perpendicular lines.

More formally, let DF be the domain of all possible dataflow instances, where $\emptyset \in DF$ is a special empty dataflow. Also, let $x : DF \rightarrow DF$ be a function that transforms a dataflow instance into another, and $\mathscr{D}$ be the set of all such functions. A vistrail node corresponding to a dataflow d_n is constructed by a sequence of actions, where each $x_i \in \mathscr{D}$:

$$d_n = x_n \circ x_{n-1} \circ \cdots \circ x_1 \circ \emptyset$$

In what follows, we use the following notation: we represent dataflows as d_i, and if a dataflow d_j is created by applying a sequence of actions on d_i, we say that $d_i < d_j$ (*i.e.,* the vistrail node d_j is a descendant of d_i). A vistrail VT can be thought of as a set of actions x_i that induces a set of visualizations d_i. The actions are partially ordered. In addition, $\emptyset \in VT$, $\forall x \in VT, x \neq \emptyset, \emptyset < x$ and $\nexists x \in VT, x < \emptyset$.

Internally, the system manipulates a vistrail using an XML representation. An excerpt of the vistrail XML schema is shown in Figure 7. For simplicity of the presentation, we only show subset of the schema and use a notation less verbose than XML Schema. A `vistrail` has a unique id, a name, an optional annotation, a set of actions, and a set of macros. Each `action` is uniquely identified by a timestamp (`@time`), which corresponds to the time the action was executed. Since actions form a tree, an action also stores

```
type Vistrail = vistrail [ @id, @name, Action*,
                           Macro*, annotation? ]

type Action =
 action [ @parent, @time, tag?, annotation?,
          (AddModule|DeleteModule|ReplaceModule|
   AddConnection|DeleteConnection|SetParameter)]

type Macro =
 macro [ @id, @name, Action*, annotation?]
```

Figure 7: Excerpt of the vistrail schema.

the timestamp of its parent (`@parent`). The different actions we have implemented in our current prototype include: adding, deleting and replacing dataflow modules; adding and deleting connections; setting parameter values. A `macro` contains a set of actions which can be reused inside a `vistrail` (see Section 5) To simplify the retrieval of particularly interesting versions, a node may have a name (the optional attribute `tag` in the schema) as well as annotations.

5. DATA EXPLORATION THROUGH WORKFLOW MANIPULATIONS

Capturing provenance by recording user interactions with the system has benefits both in uniformity and compactness of representation. In addition, it allows powerful data exploration operations through direct manipulation of the version tree. In this section, we discuss three applications enabled by these manipulations. First, we show that stored actions lend themselves very naturally to reuse through an intuitive macro mechanism. Then, we describe a bulk-update mechanism that allows the creation of a large number of visualizations of an n-dimensional slice of the parameter space of a dataflow. Finally, we discuss how users can easily create visualizations by analogy.

5.1 Reusing Stored Provenance

Visualization pipelines are complex and require deep knowledge of visualization techniques and libraries such as for example VTK [28]. Even for experts, creating large pipelines is time-consuming. As complex visualization pipelines contain many common tasks, mechanisms that allow the re-use of pipelines or pipeline fragments are key to streamlining the visualization process.

Figure 9 shows a simplified example of a dataflow that reads a data, applies a colormap, creates a scalar bar, and finally, adds some text before rendering the images. Since the last three steps

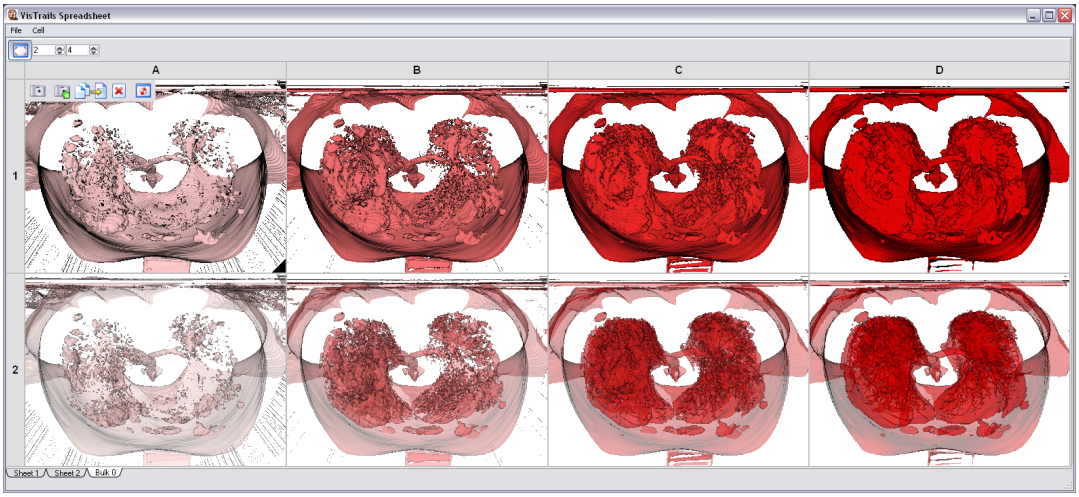


Figure 8: Vistrail Spreadsheet. Parameters for a vistrail loaded in a spreadsheet cell can be interactively modified by clicking on the cell. Cameras as well as other vistrail parameters for different cells can be synchronized. This spreadsheet contains visualizations of MRI data of a lung and was generated procedurally with the use of bulk changes. The horizontal axis varies isosurface value while the vertical axis explores different opacities.

are needed for each time step, they can be made into a macro reused in all these pipelines.

The sequence of actions stored in a vistrail leads to a very natural representation for a macro. Inserting a macro in a vistrail node d_i is conceptually very simple: the macro actions are applied to the selected workflow corresponding to the vistrail node d_i. More formally, a macro m can be represented as a sequence of operations

$$x_j \circ x_{j-1} \circ \cdots \circ x_i$$

To apply this macro to a vistrail node d_i, we *compose* the two sets of actions:

$$(x_j \circ x_{j-1} \circ \cdots \circ x_i) \circ d_i$$

There are several possible ways of defining a macro. The simplest is to do so directly on the version tree, by specifying a pair of versions d_j and d_i, where $d_i < d_j$, and define the macro as the sequence of actions that takes d_i into d_j. Another way of defining a macro is by interactively selecting a set of modules and connections in a given dataflow, so that these modules can be re-created in a different pipeline. The problem here is that there is a mismatch between the action-oriented model of a vistrail and the dataflow representation. Whereas intuitively a macro corresponds to a set of modules and connections in a dataflow, since dataflows are not stored directly, it is necessary to identify the actions in a vistrail that create and modify the modules and connections selected. However, between the first action x_i and the last action x_j, there might be actions that change parts of the pipeline that were not selected. These potentially *irrelevant* actions must be removed. In the current implementation, we perform a simple analysis and remove actions that are applied to modules that are not created between x_i and x_j. More complex, application-dependent analyses are possible.

The final important feature of macros is the *context*. When a user defines a macro, some of the actions will create new pipeline modules. Even though some of the connections will be between modules created inside the macro, some other will connect to previously existing modules. When the macro is to be applied to a different pipeline, the set of external modules to which the macro modules will connect to will change. In VisTrails, the user is prompted to choose the right modules to connect the macro to. To help the user identify the matching modules, we store, together with the macro, the external modules from the original pipeline.

5.2 Scalable Derivation of Visualizations

The action-oriented model also leads to a very natural means to *script* dataflows. In what follows, we describe a bulk-update mechanism that leverages this model to greatly simplify the creation of a large number of related visualizations. To simplify the exposition, we restrict the discussion to explorations that involve only combinations of different values for an n-dimensional slice of the parameter space of a dataflow. Since parameter value modifications and dataflow modifications are captured uniformly by the action-based provenance, a similar mechanism can be used to explore spaces of different dataflow definitions.

As discussed in Section 4, a dataflow d consists of a sequence of actions applied to the empty dataflow:

$$x_k \circ x_j \circ \cdots \circ x_i \circ \emptyset$$

The *parameter space* of a dataflow d, denoted by $P(d)$, is the set of dataflows that can be generated by changing the value of parameters of d. From $x_k, \cdots, x_i$, we can derive $P(d)$ by tracking *addModule* and *deleteModule* actions, and knowing $P(m)$, the parameter space of module m, for each module in the dataflow. Each parameter can then be thought of as a *basis vector* for the parameter space of d. It is easy to see that a set of *setParameter* actions on different dataflow parameters is the specification of a vector of this parameter space. Dataflows spanning an n-dimensional subspace of $P(d)$ are generated as follows:

$$setParameter(id_n, value_n) \circ \cdots \circ setParameter(id_1, value_1) \circ d$$

Bulk updates greatly simplify the exploration of the parameter space for a given task and provide an effective means to create a large number of visualizations. A composite visualization constructed with the bulk-update mechanism is shown in Figure 8.

Note that since VisTrails identifies and avoids redundant operations, dataflows generated from bulk changes can be executed efficiently—the operations that are common to the set of affected dataflows need only be executed once.

After a bulk update, if the user decides to keep a certain parameter setting, he has the choice to store it in the vistrail. This is easy to do, since the representation for a specific dataflow within a bulk change is the same as a regular dataflow inside a vistrail.

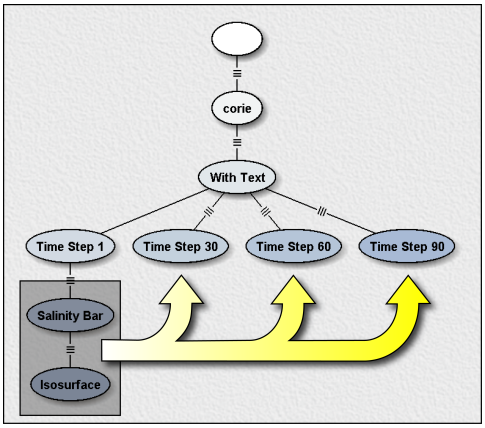


Figure 9: A vistrail macro consists of a sequence of change actions which represent a fragment of a dataflow. Here we show an example of a series of actions that occur on one version that can be applied to other similar versions using a macro.

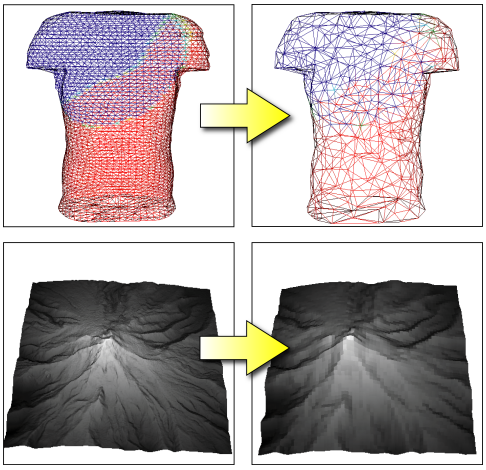


Figure 10: An example of analogy-based visualization. The simplification process that takes the top left image to the top right can be used to take the bottom left image to the bottom right.

5.3 Analogy-Based Visualization

Analogical reasoning is used to relate an inference or argument from one particular context to another. The problem of determining an analogy can be summarized as follows: if b is related to a in some way, what is the object related to c in this same way? Analogy provides a powerful means of reasoning and exploring a problem domain, and it is especially useful in visualization. Figure 10 illustrates an example of visualization by analogy. The same simplification applied to the torso dataset (top row) is applied the Mount Hood model (bottom row). By examining two images, however, it may be hard (and sometimes impossible) to precisely define their relationship.

The vistrail action-based model makes it possible to precisely define the relationship between two images. Given two dataflows d_a and d_b such that $d_a < d_b$, the relationship $R(d_a, d_b)$ consists of all the actions applied to d_a to derive d_b:

$$d_b = x_k \circ x_{k-1} \circ \cdots \circ x_j \circ d_a$$

Now, given a dataflow d_c, to create d_d by analogy with $R(d_a, d_b)$, we must first translate the actions to the context of d_c. Notice that this is equivalent to defining a macro starting at d_a and ending at d_b, and applying it at d_c.

In complex problems, such as cancer treatment planning (see Section 2.2), a set of different visualizations is often necessary to help physicians identify pathologic tissue. When the physician finds a favorable set of parameters for one visualization, he will likely need to change other related visualizations in the same way. Instead of having to identify the relevant operations, which may have taken place over a long period of time, he can tell the system to automatically infer, *by way of analogy*, which changes are needed. This makes it possible for non-experts to derive complex visualizations.

6. A COLLABORATIVE PLATFORM FOR DATA EXPLORATION THROUGH VISUALIZATION

Data exploration through visualization is a complex process that requires close collaboration among domain scientists and visualization experts [15]. Thus, adequate support for collaboration is key to fully exploit the benefits of visualization. In this section, we describe how we use the action-based provenance mechanism to allow several users to collaboratively, and in a distributed and disconnected fashion, modify a vistrail—collaborators can exchange patches and/or synchronize their vistrails.

Vistrails and Monotonicity. A distinctive feature of the VisTrails provenance mechanism is *monotonicity*: nodes in the vistrail history tree are never deleted or modified—once pipeline versions are created, they never change. Having monotonicity makes it possible to adopt a collaboration infrastructure similar to modern version control systems (*e.g.,* GNU Arch, BitKeeper, DARCS). The idea is that every user's local copy can act as a repository for other users. This enables scientists to work offline, and only commit back changes they perceive as relevant. Scientists can also exchange patches and synchronize their vistrails.

As we discuss below, the main challenge in providing this functionality lies in keeping consistent action timestamps—the global identifiers of actions within a vistrail (Section 4). Intuitively, in a distributed operation, two or more users might try to commit visualizations with the same timestamp, and a consistent relabeling must be ensured.

Synchronizing Vistrails. We call *vistrail synchronization* the process of ensuring that two repositories share a set of actions (or, equivalently, visualizations). Figure 11 gives a high-level overview of the synchronization process. Because of monotonicity, to merge two history trees, it suffices to add all nodes created in the independent versions of a vistrail. In what follows we describe an example scenario that illustrates the issues that must be addressed in vistrails synchronization.

Suppose user A has a vistrail which is checked out by both users B and C. User B creates a new visualization, represented by a sequence of actions with timestamps $\{10, 11, 12\}$. Unbeknownst to user B, user C also creates a new visualization, which happens to have overlapping timestamps: $\{10, 11, 12, 13\}$. User C happens to commit its visualization before user B, so when B decides to commit this changes, there will already be actions with his timestamps. The only solution is for A to provide a new set of action timestamps, which A knows to be conflict free (say, $\{14, 15, 16\}$), and report these back to B. The problem appears simple, except B

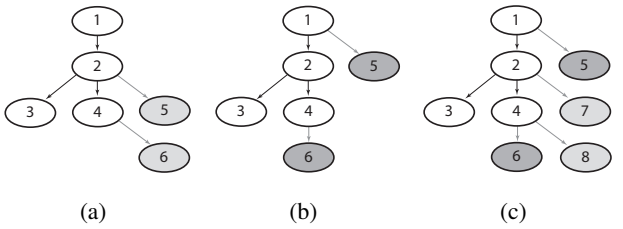


Figure 11: Synchronizing vistrails. When users and collaborate in a distributed fashion (subfigures (a) and (b)), they might create actions with the same timestamp. When these are committed to the parent repository, some timestamps have to be changed (subfigure (c)).

might himself have served as a repository for user D, who checked out $\{10,11,12\}$ before B decided to commit. We illustrate this in Figure 12. If B ever exposed his changed timestamps, a cascade of relabellings might be necessary. Worse than that, D might be offline, or depending on the operation mode, B might even be unaware of D's use of the vistrail.

Our solution is based on a simple observation. Action timestamps need to be unique and persistent, *but only locally so*. In other words, even if user A exposes his actions to B, as a certain set of timestamp values, there is no reason for B to use the same timestamps. The problem lies exactly when actions created by user B have timestamps that may be changed in the future, when committed to A. To avoid that, we introduce what we call *relabeling maps*, a set of bijective functions $f_i : \mathbf{N} \rightarrow \mathbf{N}$. Each user keeps a relabeling map whose preimage is the set of timestamps given by the parent vistrail i, and whose image is a local set of timestamps which will be exposed in case its vistrail is used as a repository. When the user commits a set of actions, the parent vistrail might provide a new set of timestamps (more specifically, the parent creates new entries on its own relabeling map, and exposes new timestamps). The child vistrail's relabeling map then *only changes the preimage*. In the previous paragraph's example, part of B's relabeling map preimage goes from $\{10,11,12\}$ to $\{14,15,16\}$, but the image stays the same. If we call f_B the old relabeling map, and f'_B the new one, then $f_B(10) = f'_B(14)$, $f_B(11) = f'_B(15)$ and so on. Notice that in this way, it does not matter what B's relabeling map is. The important feature is that its image does not change when B commits back to A. Since D's repository only depends to the image of f_B, D will never be affected by any actions of B, a property essential for scalable distributed operation.

Failure mode. The distribution model of vistrails allows for operation under peer failure. Using the above example, assume User B's hard drive fails, losing his vistrail repository. Even though the local changes are lost, some of the data might be available in User D's vistrail. In failure mode, we allow D to commit changes directly to A (or any other repository). Even though this makes it possible to prevent data loss, some redundancy becomes inevitable. Since User B's relabeling map has been lost, it is impossible to know the mapping between User D and User A's timestamps. We simply assume, then, that all actions User D wants to commit are new. The most important feature of this operation mode is that it does not violate *monotonicity*. User A's vistrail is still valid, User C might still use User A's vistrail, and User D will simply receive a completely new preimage for its relabeling map. The most important feature of the scheme is that users that have checked out User D's vistrail will not be aware of User B's failure.

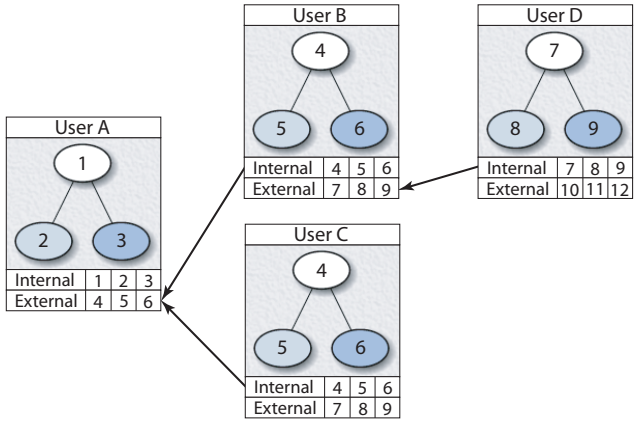


Figure 12: Synchronizing vistrails through *relabeling maps*. Even though its local timestamps might change on commits, each vistrail exposes locally consistent, unchanging timestamps to the world, ensuring correct distributed behavior.

7. EVALUATION AND CASE STUDIES

To evaluate the utility VisTrails in a practical setting, we describe how our system simplifies the visualization processes used in the motivating examples described in Section 2.

Streamlining the CORIE Visualization Processes. The process by which Professor Baptista and his staff create visualizations with custom-built scripts for the CORIE project is both time-consuming and error-prone. VisTrails not only helps streamline this process, but it also facilitates collaboration among members of Baptista's group. Because detailed provenance is captured, every image can be easily reproduced and modified. For example, if Dr. Baptista wants to regenerate a series of figures using more recent forecasts, he can query the repository for the desired vistrails and easily replace the old data sets with the new ones. With VisTrails, he has the ability to steer his own simulations and explore the data.

Another important feature of VisTrails is the ability to perform comparative visualization. The VisTrails Spreadsheet is an efficient tool for comparing multiple visualizations. For example, a new forecasting technique may be compared to an old one by showing different visualizations side-by-side in the spreadsheet where they can be interactively rotated, zoomed, and probed. Furthermore, with the use of *bulk updates* (see Section 5), multiple visualization techniques or parameter values can be easily generated with a simple interface to allow the scientist to explore the data and find a desired image quickly. Note that because VisTrails employs a caching mechanism, which avoids redundant computations [3], comparative visualization can be performed efficiently, at interactive speeds.

Replacing the Laboratory Notebook in Cancer Treatment Planning. The documentation process required to generate a set of images or movies for radiation oncology treatment planning is difficult and tedious. By capturing both data and dataflow provenance, VisTrails provides a convenient (and automatic) alternative to maintaining a laboratory notebook.

Using VisTrails, the visualization specialist can easily explore the parameter space of a visualization, incorporate new suggestions quickly, and regenerate the series of original images automatically. As an example, a clinician looking for a lung tumor often prefers visualizing the full breathing cycle as an animation of the different time steps for both the pathological and normal tissue. Using

parameter exploration through bulk changes, the visualization specialist can automatically show different time steps and parameters using one or more pipelines in the spreadsheet and compose a video with the push of a button. This allows animations of different tissue types to be contrasted simultaneously.

The VisTrails support for collaborative visualization allows simpler interaction among physicians and visualization specialists—they can work on shared vistrails and exchange patches. The simplified process to create visualizations, in particular the ability to create visualizations by analogy (see Section 5), makes it possible for the physicians themselves to explore the data and try different parameter settings.

8. RELATED WORK

The first implementation of VisTrails [3] only tracked provenance of visualization products: for a given visualization, it stored the steps and parameters that led to the visualization. In [6], we introduced the notion of action-based provenance and how that captures the evolution of dataflows. In this paper, we give a detailed description of the vistrails data exploration infrastructure, and show how the action-based provenance can be used to implement features that greatly simplify and streamline the scientific discovery process.

Visualization Systems. Several systems are available for creating and executing visualization pipelines [13, 16, 24, 28, 35]. Most of these systems use a dataflow model, where a visualization is produced by assembling visualization pipelines out of basic modules. They typically provide easy-to-use interfaces to create the pipelines. However, as discussed above, these systems lack the infrastructure to properly manage a large number of pipelines; and often apply naïve execution models that do not exploit optimization opportunities. The solutions we propose in VisTrails can be easily integrated with these systems.

Comparative Visualization. The use of spreadsheets for displaying multiple images was proposed in previous works. Levoy's Spreadsheet for Images (SI) [20] is an alternative to the flow-chart-style layout employed by many earlier systems which use the dataflow model. SI devotes its screen real estate to viewing data by using a tabular layout and hiding the specification of operations in interactively programmable cell formulas. The 2D nature of the spreadsheet encourages the application of multiple operations to multiple data sets through row or column-based operations. Chi *et al.* [9] apply the spreadsheet paradigm to information visualization in their Spreadsheet for Information Visualization (SIV). Linking between cells is done at multiple levels, ranging from object interactions at the geometric level to arithmetic operations at the pixel level. The difficulty with both SI and SIV is that they fail to capture the history of the exploration process, since the spreadsheet only represents the latest state in the system.

The Vistrail Spreadsheet supports concurrent exploration of multiple visualizations. The interface is similar to the one proposed by Jankun-Kelly and Ma [14], and it provides a natural way to explore a multi-dimensional parameter space (Section 5). The action-based provenance mechanism makes the vistrail model especially suitable to be used in such an interface. Users can change any of the parameters present in a dataflow and create new dataflow versions; and they can also synchronize different views over a set of parameters—changes to this parameter set are reflected in related vistrails shown in different cells of the spreadsheet.

Scientific Workflows. In recent years, there has been a growing interest in scientific workflows, as can be evidenced from a number of events (*e.g.,* [12,29,30]) and a fast growing literature on the topic (*e.g.,* [22, 23, 32]).

Although VisTrails was originally designed to support data exploration through visualization, ideas developed in the context of VisTrails have been successfully applied to scientific workflow systems in different domains. For example, the VisTrails data provenance mechanism is being used in the Emulab testbed, to track revisions of network security experiments [10, 11]; and algorithms developed for VisTrails have also been used in Kepler, a general scientific workflow system [23]. Note that, our goal in this project is not to build yet another scientific workflow system. Instead, the focus of our research is on developing general techniques and algorithms, and novel functionalities that support and streamline the data exploration process.

Data Provenance and Workflow Evolution. The issue of provenance in the context of scientific workflows has received substantial attention recently. Most works, however, focus on data provenance, *i.e.,* maintaining information of how a given data product was generated [31]. This information has many uses, from purely informational to enabling the re-generation of the data product, possibly with different parameters. However, while solving a particular problem, scientists often create several variations of a workflow in a trial-and-error process. These workflows may differ both in the parameter values used and in their specifications. If only the provenance of individual data products is maintained, useful information about the relationship among the workflows is lost. In addition, since a lot of expert knowledge is involved in the exploratory process, the change history of both data (e.g., input parameters) and processes contains important knowledge that can potentially be extracted and re-used to solve an array of problems. To the best of our knowledge, VisTrails is the first system to provide support for tracking workflow evolution.

Kreuseler *et al.* [18] proposed a history mechanism for exploratory data mining. They use a tree-structure, similar to a vistrail, to represent the change history, and describe how undo and redo operations can be calculated in this tree structure. They describe a theoretical framework that attempts to capture the complete state of a software system. In contrast, in our work, we only track the evolution of the dataflows and this allows for the much simpler action-based provenance mechanism described in Section 4.

9. CONCLUSION

In this paper we propose a new infrastructure for streamlining data exploration through visualization. The infrastructure leverages a new action-based provenance mechanism to provide useful features that allow effective and collaborative exploration of visualizations over large parameter spaces.

The provenance information captured by VisTrails can be used to augment existing scientific data repositories with the process that scientists go through to generate and analyze data. For example, a scientist can publish the vistrail used to generate the images in a paper. This has obvious benefit of allowing scientific results to be reproduced. In addition, since a lot of expert knowledge is involved in the exploratory process, having this information creates the opportunity for the development of mining techniques that extract knowledge, in the form of exploratory patterns, which can be used to solve an array of problems.

We are currently applying VisTrails to a number of different problem areas, from environmental sciences and medical imaging, to computer-aided drug design and discovery. Our initial experiences have confirmed that the VisTrails data exploration infrastructure greatly simplifies the scientific discovery process, and that it

indeed allows scientists to more effectively explore vast amounts of data. This indicates that appropriate management of both data and processes involved in visualization has the potential to substantially increase the impact of visualization in scientific discovery.

It is worthy of note that although VisTrails was originally designed to support data exploration through visualization, ideas developed in the context of VisTrails have been successfully applied to scientific workflow systems in different domains. For example, the VisTrails data provenance mechanism is being used in the Emulab testbed, to track revisions of network security experiments [10]. Algorithms developed for VisTrails have also been used in Kepler, a general scientific workflow system [23].

10. VIDEO OVERVIEW

Our submission includes a narrated video that demonstrates several of the features discussed in this paper. We encourage the reviewers to see the video. The video shows a user interacting with the VisTrails system in real time. It illustrates, among other things: the transparent, action-based provenance capture; the application of macros to reuse parts of the provenance in different situations; and scalable data derivation through the bulk update mechanism.

Unfortunately, the conference management system does not support large files and we were not able to upload the video together with the paper. The video is available at

http://www.sci.utah.edu/~vgc/vistrails/videos

and it can also be obtained from Dr. Gustavo Alonso, the PC chair for the VLDB 2006 IIS track.

Acknowledgments. Professor António Baptista (Oregon Health & Science University) has provided us valuable input for the system design. We thank him for letting us use CORIE as a testbed for the development of VisTrails. We thank Dr. George Chen (Massachusetts General Hospital/Harvard University) for providing us the lung datasets, and Erik Anderson for creating the lung visualizations. This work was partially supported by the National Science Foundation under grants IIS-0513692, CCF-0401498, EIA-0323604, CNS-0541560, and OISE-0405402, the Department of Energy, an IBM Faculty Award and a University of Utah Seed Grant.

11. REFERENCES

[1] A. Ailamaki, Y. E. Ioannidis, and M. Livny. Scientific workflow management by database management. In *Proceedings of SSDBM*, pages 190–199. IEEE Computer Society, 1998.

[2] A. Baptista, T. Leen, Y. Zhang, A. Chawla, D. Maier, W.-C. Feng, W.-C. Feng, J. Walpole, C. Silva, and J. Freire. Environmental observation and forecasting systems: Vision, challenges and successes of a prototype. In *Conference on Systems Science and Information Technology for Environmental Applications (ISEIS)*, 2003.

[3] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *Proceedings of IEEE Visualization*, pages 135–142, 2005.

[4] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML query language. W3C Working Draft, June 2001.

[5] A. Brown, M. Fuchs, J. Robie, and P. Wadler. XML Schema: Formal description, 2001. W3C Working Draft.

[6] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. Managing the evolution of dataflows with vistrails *(Extended Abstract)*. In *IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow)*, 2006. To appear.

[7] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. VisTrails: Visualization meets Data Management. In *Proceedings of ACM SIGMOD*, 2006. Demo description. To appear.

[8] E. H. Chi, P. Barry, J. Riedl, and J. Konstan. A spreadsheet approach to information visualization. In *Proceedings of IEEE Information Visualization Symposium*, pages 17–24, 1997.

[9] E. H. Chi, P. Barry, J. Riedl, and J. Konstan. Principles for information visualization spreadsheets. *IEEE Computer Graphics and Applications*, 18(4):30–38, 1998.

[10] E. Eide, T. Stack, L. Stoller, J. Freire, and J. Lepreau. Integrated scientific workflow management for the emulab network testbed. In *Proceedings of USENIX*, 2006. To appear.

[11] The Emulab Network Emulation Testbed. http://www.emulab.net.

[12] e-Science Grid Environments Workshop. http://www.nesc.ac.uk/esi.

[13] IBM. OpenDX. http://www.research.ibm.com/dx.

[14] T. Jankun-Kelly and K. Ma. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287, 2001.

[15] C. Johnson, R. Moorhead, T. Munzner, H. Pfister, P. Rheingans, and T. S. Yoo. *NIH/NSF Visualization Research Challenges Report*. IEEE, 2006.

[16] Kitware. Paraview. http://www.paraview.org.

[17] Kitware. The Visualization Toolkit. http://www.vtk.org.

[18] M. Kreuseler, T. Nocke, and H. Schumann. A history mechanism for visual data mining. In *Proceedings of IEEE Information Visualization Symposium*, pages 49–56, 2004.

[19] E. A. Lee and T. M. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[20] M. Levoy. Spreadsheet for images. In *SIGGRAPH*, pages 139–146, 1994.

[21] M. Levoy, H. Fuchs, S. Pizer, J. Rosenman, E. L. Chaney, G. W. Sherouse, V. Interrante, and J. Kiel. Volume rendering in radiation treatment planning. In *Proceedings of the First Conference on Visualization in Biomedical Computing*, May 1990.

[22] B. Ludaescher and C.Goble. Special section on scientific workflows. *ACM SIGMOD Record*, 34(3), Sept. 2005.

[23] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 2005.

[24] S. G. Parker and C. R. Johnson. SCIRun: a scientific programming environment for computational steering. In *Supercomputing*, page 52, 1995.

[25] C. A. Pelizzari and G. T. Y. Chen. Volume visualization in radiation treatment planning. *Critical Reviews in Diagnostic Imaging*, 41(6):379–364, 2000.

[26] The EU Provenance Project. http://twiki.gridprovenance.org/bin/view/Provenance.

[27] D. Roundy. Darcs. http://abridgegame.org/darcs.

[28] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics*. Kitware, 2003.

[29] IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow 2006). http://www.cc.gatech.edu/ cooperb/sciflow06.

[30] Scientific Data Management Framework Workshop. http://sdm.lbl.gov/arie/sdm/SDM.Framework.wshp.htm.

[31] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[32] E. Stolte, C. von Praun, G. Alonso, and T. R. Gross. Scientific data repositories: Designing for a moving target. In *Proceedings of ACM SIGMOD*, pages 349–360, 2003.

[33] The Taverna Project. http://taverna.sourceforge.net.

[34] The Triana Project. http://www.trianacode.org.

[35] C. Upson et al. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[36] J. van Wijk. The value of visualization. In *Proceedings of IEEE Visualization*, 2005.

[37] Extensible Markup Language (XML). http://www.w3.org/XML.

[38] XML path language (XPath) 2.0. http://www.w3.org/TR/xpath20.