This is a sample progress report. This project was started at VisTrails, Inc. in 2007 by Callahan et al. [1] and work continued through 2012. Some details have been altered for educational purposes and this document should not be used as documentation for the referenced project.

1 Introduction

As computing proliferates to many different fields, it becomes easier for everyone to do more of their work in a computational setting. This often leads to more efficient work, but with more work comes the problem of remembering and recalling many more steps followed. Provenance-enabled applications track exactly how a particular result was generated: the steps taken, any data used, and when the work was done. In this project, we take ParaView, a complex visualization tool with a graphical user interface, and add provenance capture capabilities to it.

While scientific workflow systems and other provenance-aware systems provide general frameworks for capturing provenance, ParaView has its own involved graphical user interface. Thus, trying to fit its capabilities into an existing system would be a complex process and would likely limit its functionality. We instead plan to instrument ParaView itself with provenance-capture capabilities by making use of the undo subsystem. This allows a meaningful provenance that can also be understood at the correct level of granularity for users. In addition, we intend to allow users to interact with the provenance directly through a version tree, allowing users to access past states without saving multiple versions of documents [2].

2 Related Work

While the term provenance has not always been used, people have been capturing sequences of actions that capture their work for a while. Scientists record the exact items and measures used in experiments as well as the entire sequence of steps for their work. Businesspersons create protocols to make sure work is accomplished according to rules and regulations, often requiring documented proof that each step was accomplished. In computing, undo stacks allow access to recent work, version control systems persist versions of work, and some tools store step histories (e.g. bash history). This project aims to continue this trend by bringing a more useful, automated provenance to ParaView. ParaView is an open-source visualization application that is built on the Visualization Toolkit (VTK). It has extensive support for 3D visualization and can be run on desktop machines or supercomputers [4].
Figure 1: Captured provenance from the ParaView Provenance Plugin. The provenance information is represented as a tree (shown on the right) that reflects the different paths a user (or group of users) followed. Each node in the tree corresponds to an application state. In this example, a node represents the sequence of steps followed to derive a visualization. An edge between two nodes encodes a set of actions applied to the parent to obtain the child node. The provenance information can also be presented as a list of states (shown on the left) that can be browsed and queried.

3 Design

There are three main aspects of the design: capturing provenance, replaying provenance, and the provenance interface. The capture and replay systems are linked in that the data captured must be interpretable by the replay system. Thus, we choose to keep each provenance entry as a set of state changes with metadata containing the user, time, and parent action. We use ParaView’s undo serialization mechanisms along with its “Save State” functionality to capture provenance. For replay, we use “Load State” as well as pushing changes onto the undo stack. Note that the saved states allow us to move to particular versions more efficiently than replaying every action on the undo stack.

The provenance interface is composed of a user-selectable tree. Each node will be labeled with a short tag describing the type of action (e.g. “Changed isovalue”) that can be overridden with a user-defined tag. Edges link dependent actions. To show large version trees, we compress straight paths (paths where each node is untagged and has a single parent and child) into a single edge. Any new action in the application will generate a new node and edge to the parent node it was derived from. A user will be able to select any node to return that state (see Figure 1).

4 Implementation

This project requires a user to install ParaView and the provenance plugin we will build. We have targeted ParaView 4.2 for this work. We use the VisTrails SDK [5] to store, read, and display the provenance data. Because the SDK requires Qt, we have used Qt 4.8.4. The final plugin should be cross-platform but our development has been on Mac OS X.
5 Results

[We plan to add details showing examples of the provenance based on the ParaView tutorials.]

6 Discussion

While we expected to use the undo and redo actions to move between program states, it turned out that certain actions modified the global state of the program and were not captured in the actions. For this reason, we needed to use the full ParaView state for certain actions. This adds to the size of the provenance, but we decided these actions were necessary to store as provenance. In the future, we may work to find workaround actions that may more efficiently represent this state.

A second issue we encountered was the granularity of the provenance. When a user changes the same parameter multiple times, we capture each change as a separate action. It may make sense to chunk these actions as suggested by others (e.g. [3]) when the changes are made rapidly.

7 Project Progress

- [10/2-10/20] Work on capturing raw provenance from ParaView. Completed. We have captured raw provenance from ParaView’s undo stack in the protobuf format. Each action is tagged with a user and date for debugging, and the application-specific data is the protobuf data. In addition, we capture full program states at timed intervals using the “Save State” functionality.

- [10/20-11/5] Work on replaying raw provenance in ParaView. Completed. We are able to push actions (captured in the protobuf format) onto the ParaView undo stack and trigger the state to be updated. In addition, “Load State” can be used to pull in full program states. There are a few issues with certain actions types where pushing onto the undo stack does not work so we used the full state in those instances.

- [11/6-11/12] Migrate work to use VisTrails SDK format for provenance. In Progress. We need to ingest the data we are capturing into the VisTrails SDK format. This is still in progress because we ran into an issue with serializing the actions into a format that is compatible with the SDK.


- [11/24-12/2] Write project report. In Progress. We have put some of the background material into this progress report, including some related work.

References


