Toward User Interface Migration for Scientific Visualization

Cornelius Toole, Jr.1,2, Brygg Ullmer1,2
Kexi Liu1,2, Chris Branton1,2, Rajesh Sankaran1,3
1Computer Science Department, Louisiana State University
2Center for Computation and Technology, Louisiana State University
3Electrical and Computer Engineering Department, Louisiana State University

Abstract: Visualization is a key component of the scientific investigation process, but different contexts (e.g. varying numbers, locales and expertise of users, heterogeneous infrastructure components, etc.) require different toolsets for workflow management. We focus on interactive techniques for engaging visualization, especially those that emphasize physical interaction as embodied in tangible and gestural interfaces. In building tangible user interfaces for several software platforms with diverse interaction devices, we face considerable complexity. Here we present preliminary work toward a toolkit with mechanisms for generalizing input from diverse interaction device implementations, as well as generalizing control of a generic visualization model. A key feature of this toolkit is ability to build applications for executing a visualization workflow using multiple user interface implementations with minimal code changes. The ability to easily migrate analysis workflows between user interfaces moves us toward a future in which users can select the right interactive toolset for a given context.

Keywords: human computer interaction, visualization, tangible interaction

1. Introduction

Visualization is a key component of the scientific investigation process. And while visualization community has contributed powerful tools for seeking insight from scientific data, no one visualization tool or interaction modality is appropriate for all situations. Currently, the state of the art in user engagement of visualization is fractured. If a developer wants to provide users with software that supports execution of workflows that migrate across visualization software platforms, she has to write against multiple architectures. To the extent there are application-programming interfaces (APIs) to common functionality from multiple diverse implementations, developers can use these interfaces to select the capability provider for the current job.

We focus on the user interaction aspects of this problem space. One opportunity we see is for the ability to provide common user interfaces to diverse visualization software platforms. Previous work by our group pursued this in the development of a tangible user interface that could drive visualization pipeline instances simultaneously [1]. This work is predicated on the idea that although many visualization applications are based on different architectures and internal models, many of them are based on similar general models of visualization (e.g. data flow pipeline or visualization network). Common visualization user interfaces enable users to execute common visualization tasks (e.g. loading data, invoking visualization pipelines for typical filters, changing view parameters, traversing temporal data) across platforms without requiring them to know each of the platform specific menu hierarchies or command languages.
But given the ability to decouple visualization functionality from its user interface facade, we can exploit interactive diversity in interesting ways. For instance, tangible interaction approaches offer strengths in supporting multi-user collaborative work. Some of those strengths may also be possessed by interfaces realized with gestural touch-based surfaces than can be easily shared (e.g. iPad or MS Surface) or economically replicated (iPod touch). Just as Liu et al. demonstrated the feasibility of bringing multiple visualization applications under the control of a single interaction device, we propose that it is possible to bring several interaction modalities under an application-programming interface (API) capable of driving the same functional core.

We propose a toolkit that enables users to execute visualization workflows with multiple interactive tools. We demonstrate this with software that takes advantage of this flexibility achieves this with minimal changes to source code. This toolkit is comprised of two components: generic visualization domain model and API for which adaptors can be written to drive several visualization platforms, and a generic user interaction API and model.

2. Use Case

The use case we target centers around a generic visualization use case in which a user carries out the following tasks: (1) Loads a volumetric dataset; (2) Invokes a scalar data visualization algorithm (e.g. Marching Cubes); (3) Changes view parameters to locate a region of interest (ROI) within dataset; (4) Traverses time-steps within this data to observe this ROI over time; and (5) Records snapshots of this visualization for sharing later.

3. System Implementation

We begin our path toward integration of interaction devices independent of underlying implementation by providing an API for constructing abstract interaction objects. These abstract interaction objects can be bound to concrete interaction devices during runtime. The concrete interaction devices may be realized in several ways. Consider a rotary input device such as a knob. Our software enables users to program interaction against a knob input device whether the knob is desktop GUI widget, a mechanical or optically tagged physical object on an interactive surface. The abstract interaction devices built within this toolkit act as proxies for actual devices that comprise the concrete device.

In addition to the abstract interaction object API our solution includes a collection of software components and services for transforming user interaction data between generic and device-specific forms (e.g. fan-in/fan-out or input fusion and fission). The abstract interaction objects API is based on a formal model proposed by Card et al. that describes interaction devices as object that map state changes in its input domain to changes to changes within some other object's output domain [2].
The device manager, interactive message bus, device manager proxy, and visualization domain model controller components all comprise the flexible visualization user interaction software architecture (see Figure 1).

The device manager handles communication with physical interaction devices via device drivers, which are responsible for translation to and from device specific data formats for a particular device implementation. The device manager proxy communicates with device manager over an interaction message bus and maps communication between concrete interaction devices and abstract interaction objects. Currently the interaction message bus is based upon the Zero Message Queuing (0MQ) protocol [3], a communication library that provides a uniform API to several communication models (e.g. request-reply, publish-subscribe, peer-to-peer) over several types of channels (e.g. TCP, UDP, multicast, Unix pipes, shared memory). The visualization domain model controller components map generic visualization operations to operations specific to a particular visualization software platform. The loosely coupled, logical structure of this system grants developers flexibility in the configuring their interactive visualization application.

Figure 2 shows a code snippet in which three rotors are added to an interactive context and then bound to a visualization view parameter operation via visualization parameter controller. Toolkit users do not have to specify any particular type of device or visualization application, although they have the option to do so. All knobs exhibit a rotate behavior, and most visualization applications support view parameter operation corresponding to rotation and zooming. Our goal is for developers to be able to not care about a particular device implementation, while also being able to take advantage of unique features of different implementations. For instance, a developer may wish to program interactive manipulation of the time-step parameter via a knob that takes advantage of haptic feedback to limit the user’s input bandwidth. This is useful if the result of changing a timestep is a service requests for the loading of large amounts of data.

5. Conclusion

Currently, the toolkit described here supports integration of input from several input devices including the Griffin Powermate media knob, several Blades & Tiles mechanical-electronic interaction devices, touch-based interfaces built with touchOSC/iPhone application, and desktop-GUI-based widgets. The binding of abstract interaction
object and concrete interaction devices is achieved through publish-subscribe-based coordination. This can be exploited to enable one device to manipulate digital objects. This simplifies the tasks of simultaneously controlling multiple visualization instances. It is also possible to bind several interaction device instances to a single software instance, which simplifies multi-user interaction. Combined with the decoupling of an application’s communication semantics from its reification as channels, developers can also more easily change local interactive software into distributed software.

Future work includes the development of a visualization domain model controller. This is a generalization of the preliminary API described by Liu et al. This controller is based on the visualization parameter settings (viz p-set) formal model of visualization exploration [4]. A viz p-set encapsulates all the information needed to reproduce a visualization result. This controller will expose the internal state of a visualization application instance while adaptors translate between the generic interface and the interfaces of particular visualization software platforms. The work described here moves us close to the goal of being able to realize interactive visualization applications that can vary in the number and locality of users, the number and class of interaction devices, and the number and class of visualization software instances.

6. Acknowledgments

The current work is funded by the NSF EPSCoR CyberTools project under award #EPS-0701491. This work was also funded in part by the NSF Bridge to the Doctoral Program Fellowship and the Huel D. Perkins Doctoral Fellowship.

7. References


