VisSheet Redux: Redesigning a Visualization Exploration Spreadsheet for the Web

T.J. Jankun-Kelly and Kwan-Liu Ma
University of California, Davis
{kelly,ma}@cs.ucdavis.edu

Introduction

The exploration of complex data sets requires interfaces to present and navigate through the visualization of the data. In recent work [Jankun-Kelly and Ma 2001], we produced a visualization exploration spreadsheet to address this issue. The developed application, however, was implemented for off-line use only. For data sets on remote sites, this approach is not appropriate. Thus, a web-based version of the visualization exploration spreadsheet is needed. This abstract discusses the process of transforming the interface from an off-line to an on-line design.

From Off-line to On-line

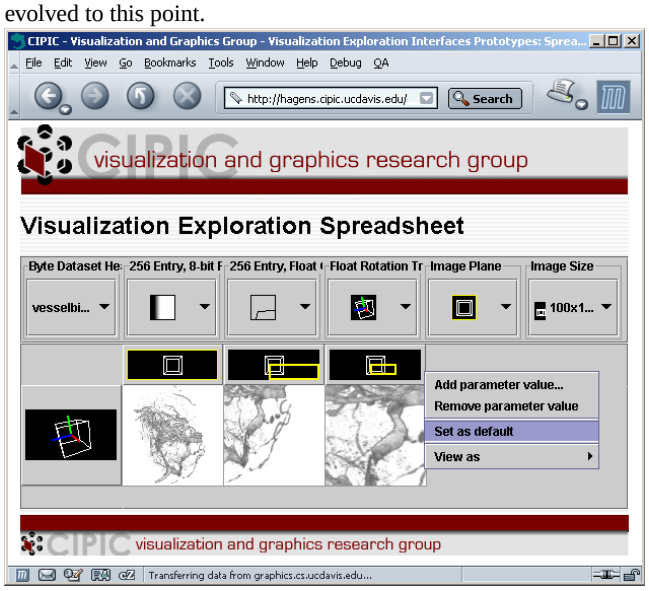
Our spreadsheet-like interface (VisSheet, for short) was designed to assist visualization exploration by providing context for where the user is in their exploration, where they have been, and suggesting where they may go next. The VisSheet addresses these tasks by providing a movable, scalable window into the visualization parameter space. By manipulating the visualization parameters, the user changes the position and size of this window. Only two visualization parameters are displayed at a time: one along the rows and another along the columns. For the non-displayed parameters, a set of default values is maintained which may be updated at run-time. Parameter values are rendered as glyphs. Cells—representing a combination of the row, column, and default parameter values—display the visualization results. By changing the the default values for non-displayed parameters or which parameters are displayed along the rows or columns, the "window" can be moved in the visualization space. Thus, the data exploration process becomes the process of manipulating the spreadsheet window through the visualization space.

The original design of the VisSheet consisted of three major components: the active View on the visualization parameter space, the Model of the visualization transform responsible for the parameter space, and a Session which records the user's process. These components were all designed to be modular. For example, one could change the type of visualization performed by replacing the visualization transform encapsulated by the Model object. This module handles the rendering requests. By replacing the Model object with a version that performed remote visualization, the application could become partially on-line. This configuration, however, was fully "web-aware": it could neither be accessed on-line nor visualize data from different servers. Thus, a redesign was performed.

There were two major goals for the redesign effort. The first was to allow the VisSheet to be "fully" on-line. By this, we mean that the application should be accessible over the Internet on a wide variety of machines. The second goal was to maintain modularity. The VisSheet is only one of the exploration interfaces being developed. It is important that the reusable components common to the other interfaces—such as the modules capturing the exploration process—could be reused. The modularity also extends to the visualization itself: Like the original VisSheet, the new version must be able to interface with different visualization transforms.

The first goal determined a series of constraints on the new design. Since the application was to be on the web, it was assumed that there would be no interaction with the local machine. In addition, it is desirable that users of the on-line VisSheet have a very low barrier of entry. Fewer constraints were due to the second goal. At the time of the VisSheet redesign, the modules forming the core of the new visualization exploration framework were also be written. Thus there was no legacy code to support in the new implementation.

The final design of the system consists of rewrites of the original three major systems. The visualization framework itself has been implemented in Python. In the new framework, an abstract, UI-toolkit independent VisSheet module has been developed; for the web VisSheet, a Java interface is provided by using Jython, a Python environment for Java. In our prototype, the VisSheet applet, written in Java and Jython, communicates to a volume visualization server written in Python with a C++ hardware accelerated renderer. This interoperation of languages attests to the modularity of the system. This system will be demonstrated, detailing both how the framework assists remote visualization and how the system evolved to this point.



Oliver Kreylos provided the texture-based volume renderer for our use.

References

JANKUN-KELLY, T. J., AND MA, K.-L. 2001. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics 7*, 3 (July/Sept.), 275–287.