SLAC-PUB-9178
March 2002

The HippoDraw Application and the HippoPlot C++ Toolkit upon which it is built

Paul F. Kunz
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94309

1. Introduction

HippoDraw is a highly interactive, document centric data visualization application. It was first implemented on NeXTStep where it was highly acclaimed by its limited user base[1]. It is now a cross-platform application with GUI components written in Java and the underlying core using the HippoPlot C++ class library.

The HippoDraw application is based on a document paradigm much like a word processor or drawing program. This idea is simple but has been used in a number of unexpected ways. The principle user interface is GUI-based. The interface is easy to use and makes it quicker for the user to display how he wants things. Its highly interactive nature gives its users a better feel for the data being displayed.

2. Overview of the Application

Figure1 shows a screen dump of a running HippoDraw application. In the center is the large window containing the drawing canvas. This is the document window. it is the focus of the user's attention. The

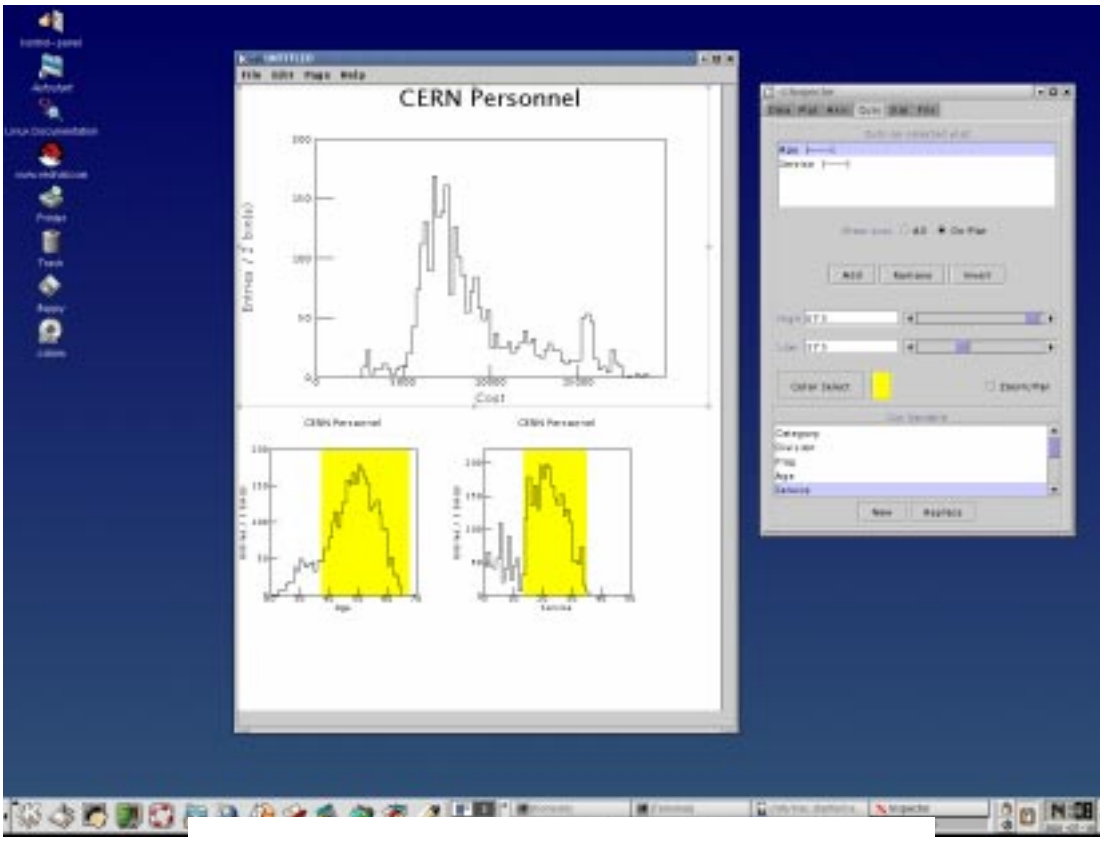

Fig.1. Screen dump of running HippoDraw application

Work supported by the Department of Energy contract DE-AC03-76SF00515.
Presented at the International Conference on Computing in High-Energy and Nuclear Physics (CHEP'01), 9/3/2001 - 9/7/2001, Beijing, China

size of this window is such that it prints as standard U.S. letter or international A4 size. A document may contain any number of pages and multiple documents can be opened at one time.

To the right of the main canvas window is the inspector panel. It is via this panel that one can changes the attributes of the graphic objects, except for resizing or moving them on the canvas. The latter attributes are changed by selecting and direct manipulation with the mouse. User input on the controls in the inspector window are applied to the selected graphic. One also uses the inspector to view the graphic object's properties.

In the menu bar on the canvas window the user will find further controls for opening files, changing fonts, grouping and alignment of graphics, *etc*. These are controls that either have only a on-off choice, or need to bring up additional panels, such as file dialog, for the user to make a choice.

The inspector panel has six sub-panels which are selected by the tabs at the top of the panel. Thus, the inspector panel is usually on the screen all the time, but its contents change to allow the user access different controls. We found this design to be better than separate panels for different set of controls because when using applications that have separate panels, we find the screen gets rapidly cluttered, with some of the panels even hiding parts of the document window. One should note from figure 1, that not only did we decide on only one control panel, but it is also small enough so it is unlikely to sit on top of the document window or exceed the size of a laptop computer's screen.

One of the tabbed panels is the "Data" panel from which one can select the kind of display to create and its binding to the n-tuple data. Nine kinds of displays have been implemented so far. The GUI of this panel is not hard wired to the kinds of displays. Rather, the GUI asks the C++ display factory instance for the list of displays and their potential bindings. The next tabbed panel selects the representations, point sizes, and color.

The "Axis" panel sets the range of the data displayed can be set by sliders or by exactly values typed in by the user. Unique to HippoDraw is a slider which controls the bin width. Changing the bin width requires that a histogram be re-accumulated. But with today's computers, . this operation is very fast. In fact, except for very large data sets, the time needed to re-accumulate and display is dominated by the time to re-draw. One can see many new displays per second even on the slowest computers in use today. Allowing one to continuously change the bin width gives a better feel of the significant of the peaks and valleys that might appear in a histogram. There is also an offset control, which moves the edges of all the bins by a fraction of the bin width. It also helps the user understand the significance of what he is seeing.

The next tabbed panel controls the adding, removing, and setting of cuts that can be applied on any display. As seen in Figure 1 when a cut is added, a histogram of the column used by the cut is displayed with a shared region showing the accepted values. The cut range is controlled by sliders and/or text fields. For each change in a cut range, the display being cut is instantly redraw. Thus, one can change the cut range smoothly and watch the effect on the target display.

The "Fits" tabbed panel allows for overlaying functions on displays and fitting them to the data. Here again, the GUI does not have hard wired into it a fixed set of functions. Rather it receives the list of available functions from a function factory C++ object. Also, the number and names of the function parameters is obtained by asking the function C++ object for it. In this way, the available functions can be extended by adding C++ function instances to the function factory.One can also do a linear sum of functions.

The final tabbed panel provides a way of adding numbers to the canvas. One such number is the total number of entries in a display. The displayed number is "live" in the sense that if the number of entries changes, say because of changes in a cut range, it updates itself. Also the Chi-squared of a function on a display and/or the function parameters can be displayed. Again, if the display data changes, the Chi squared in automatically updated. All these number "float" on top of their display so that the user can move them to a more appropriate place then the default.

3. Document paradigm

Two additional capabilities are needed to complete the document paradigm. The first is the ability to save the canvas and the state of all objects into a file that can be opened at a later date. The second is to be able to print the document. With these features, there are additional benefits, which will be described in the following sections

In applications that have commands, a script written in some language is used to re-display graphs at a later time. With HippoDraw, this need for a scripting language and its corresponding commands is eliminated. Instead, the document is stored with all the attributes of the graphs including cuts, fits, *etc*.

From their first physics laboratory course, every physicist keeps a log book that records the steps taken to reach a result. With a typical histogram viewing application, one needs to print what one sees on the screen and paste the plots into the logbook at pertinent points in the analysis chain. With HippoDraw, users have used the document file as their logbook. That is, when they were satisfied with some point along the analysis chain, but need to proceed further, they leave that plot on the canvas and create a new plot for the next step. The result is the sequence of important plots and their cuts, fits, *etc.*, is recorded in the document. We have found that users have typically saved documents of 3-7 pages in length.

Another use of commands and scripting language is to do something very repetitive, like generate 50 histograms on 50 different channels of something. A scripting language that supports looping and some rudimentary math makes such tasks much easier. This is done with HippoDraw by running the application as module in an interactive and/or scripted Python session. Python[2] is an interpreted object oriented language that is getting increasingly popular in the science community. From a Python session and/or Python script, one has the same capability to create, manipulate, and inspect displays that the GUI has. For example, one can fit functions to a histogram and read back the parameters of the fit. The graphic objects on the canvas make no distinction on whether messages are coming from Python or from the GUI. To make this possible, the Java implementation of Python, Jython[3], is used.

One of the uses of commands and scripting languages in other applications is to generate the same set of plots, cuts, *etc*. on n-tuples that have the same format but correspond to different data sets. The way this need is resolved in HippoDraw was to use a saved document as a template for many data sets. When the document is saved, by default only a reference to the n-tuple file is saved, not the data itself. Thus, if the document is opened at later date and the contents of the n-tuple has changed, then the contents of all the plots will change automatically. Likewise, a user can open one or more additional n-tuples of the same format, select all the plots, then replace the n-tuple used by them via the data inspector.

Another use of running Hippodraw under Python is when data in different formats need to be read. From a Python session and/or script, an ntuple can be created and then filled either by row or by column. Any Java package capable producing row-wise or column-wise data can be used. Two examples are the JavaFits[4] package for reading astrophysics standard FITS formatted files, and the hep.io.root package[5] for reading the data in ROOT files. Another possibility, is to read data files, massage the data, create vectors of data that can be added by row or by column to an n-tuple.

4. The HippoPlot C++ class library

The HippoDraw C++ library is a toolkit for building data visualization applications. The design of this tool kit is an attempt to decompose into abstractions the process of displaying data in various ways. Thus, there are a number of class hierarchies, each representing an aspect of the decomposition. For example, the projector hierarchy is responsible for reading data and creating the x-y points, called projected values, that are to be plotted. The simplest one is used to create a scatter plot, while a more complex one is used for a histogram.

Since projected values can be represented on a plot in many ways, there is a point representation hierarchy. A member of the plotter hierarchy requests projected values from a projector, and hands them

one by one to a point representation, which then draws the point to an abstract view class. The view class has a handful of methods. A concrete implementation of a view class interfaces to Java for drawing, but other interfaces, such as Qt, could be easily implemented.

Ten kinds of displays have been implemented so far by putting together different members of each class hierarchy. It should be relatively easy to create new kinds of displays that can be useful to both the particle physics and astronomy communities.

The toolkit is independent of the Java based GUI of the application. It could be used with a different GUI toolkit, for example with Trolltech's Qt. It may also be used for different applications.

5. Some Details

HippoDraw, and thus the HippoPlot class library, has been ported to Microsoft Windows and to Mac OS X, as well as to various versions of Linux. It compiles with stable releases of gcc from egcs-1.1.2 through gcc-3.0, Visual C++ 6.0, and Sun's CC 5.1. The standard GNU build environment consisting of autoconf, automake, and libtool has been used. The class library consists of about 100 relatively small classes. Sources are available from ftp://ftp.slac.stanford.edu/users/pfkeb/hippodraw/. A Microsoft installer file of built application is also available in the same directory.

6. Conclusions

HippoDraw has demonstrated interesting features which are unique to this application. It is predominately GUI based, but can be used within a Python session for scripting and a command-like interface. The GUI based control environment allows one to visualize and understand the data much more quickly and with far greater ease. The document-centric paradigm also proved to be very useful in ways that were not foreseen. The Python interface makes up for what can not be done solely by the GUI based controls.

Acknowledgments

The users of HippoDraw have contributed many ideas via constructive criticisms and suggestions. Amongst the most vocal were Bill Atwood and Tom Pavel. The author was aided in development of the original application by Mike Gravina, Paul Rensing, and more recently by Jeff Gould, Gilead Wurman, Oded Wurman, Stefan Bonneaud, and Matan Shacham.

References

1 Michael F. Gravina, Paul F. Kunz, T. Pavel, P. Rensing Proc. Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High-Energy and Nuclear Physics, La Londe-Les-Maures, France, Jan 13-18, 1992. World Scientific.
2 http://www.python.org.
3 http//www.jython.org.
4 http://heasarc.gsfc.nasa.gov/docs/heasarc/fits/java/v0.9/.
5 http://java.freehep.org/lib/freehep/doc/root/index.shtml.