SpecTk: a displayer for NSCL SpecTcl

D. Bazin

National Superconducting Cyclotron Laboratory, Michigan State University

East Lansing, MI 48823, USA

I. Introduction

One of the main assets of Tcl/Tk is the ability to create easy-to-use packages and libraries. In many other forms of programming, the effort necessary to reuse existing resources often outweighs the benefits, leading to frequent cycles of "reinventing the wheel". The program SpecTk described in this paper wouldn't have been written without the existence of the BLT[1] and Incr Tcl[2] packages. In fact, the inspiration for creating this program came while using the BLT package in a simpler application. Because it is aimed at displaying numerical data, this package provides most of the resources required to build an efficient, professional-looking and user-friendly displayer for the NSCL SpecTcl data analysis program[3]. The resources that are not available from the package had to be implemented in C++ because of the required speed, but here again the API provided by the BLT package proved essential to the ease of implementation. The Incr Tcl package provides an elegant and simple object-oriented framework which has the added advantage of automatically isolating class objects in their own namespaces.

This paper describes the motivation behind the SpecTk displayer program, the functionalities it provides and their purpose, as well as its overall architecture. The emphasis focusses on the ways the Tcl/Tk scripting language and its derived packages can be efficiently and easily used to implement the various components of this application. It should be noted that the author of this program is not a programmer nor a computer scientist, but a physicist in nuclear science who takes programming as a serious hobby. Although this may look like a disadvantage from a programmer's point of view, it gives the author a unique opportunity to build the best suited tools, thanks to his first-hand end-user experience acquired during daily work on experiments, as well as his interaction with students and scientists analyzing data.

---

[1] The BLT library, by G. A. Howlett

[2] The Incr Tcl library, by M. McLennan

[3] The SpecTcl Data Analysis System, R. Fox, C. Bolen, K. Orji and J. Venema, Proceedings of the 2003 IEEE Conference on Real-Time Applications of Computers in Nuclear, Particle and Plasma Physics

II. Motivation

In experimental nuclear physics like in most other science domains nowadays, observation relies primarily on indirect detection of phenomena through an ensemble of devices. The experimenter has a mental representation of the phenomena, and carries this concept through an understanding of the devices, all the way to the visualization of data returned by those devices. It is therefore of utmost importance that the last link in this chain of representation - the displayer showing the data - be accurate and meaningful. The relationship between the visualized data and the measure of the actual phenomena - what is called an "observable" in quantum mechanics - is so far reached and fragmented that extensive analysis needs to be performed on the data to before it can lend itself to an interpretation.

The primary purpose of the NSCL SpecTcl data analysis program is to establish a link between the data recorded from the devices, usually in an event-driven format, and the experimenter via visualization of this data in its raw form and/or some more sophisticated and meaningful combination or subset of it. To be able to perform this task while the experiment is taking place is crucial for diagnostic purposes, in particular when it involves extracting information in meaningful units such as distance or energy for instance. The motivation for creating a displayer for the NSCL SpecTcl program came from - as often the case in programming - frustration and struggle with the existing tool to visualize the data, and the recognition that the BLT library had most of the resources required to do a better job. In addition, the NSCL SpecTcl data analysis program being based on Tcl/Tk, it seemed natural to build a displayer also based on the same framework (the existing displayer was actually written for a previous analysis program, prior to NSCL SpecTcl), with the many associated programming advantages.

III.Design and architecture

1. Design requirements

The customary way of representing event-driven data is via the use of histograms. A histogram is a frequency distribution of one or two parameters, ordered in discrete chunks also called bins. The number of bins to display the frequency distribution can be chosen arbitrarily and only affects the precision of the representation of the parameter(s). The width of a bin corresponds to the range of parameter values for which each event will increment that same bin. The limitation to two parameters is purely technical, because an additional dimension is needed for the frequency axis, and we live in a three-dimensional world. Also, memory requirements grow exponentially with dimension. As a result, the multi-dimensional space defined by the many parameters involved in a typical experiment can only be explored through one- or two-dimensional projections. In addition, it is almost always necessary to restrict the data to subsets,

thereby involving the use of filtering objects also called gates, which test single or pairs of parameters and condition individual histograms. A displayer must therefore be able to display and manipulate all these objects in an interactive way with the data analysis program.

The design of SpecTk is based on the following requirements:

- Visualize and inspect histograms in a variety of ways
- Superimpose histograms for visual comparison
- Visualize many histograms simultaneously on pages
- Label histogram condition and axis with relevant information
- Provide easy access to large number of histograms
- Display, create and manipulate gates or regions-of-interest (ROI)
- Perform basic statistics calculations such as fits and integrals on data
- Possibility to connect to different source of data

2. Network architecture

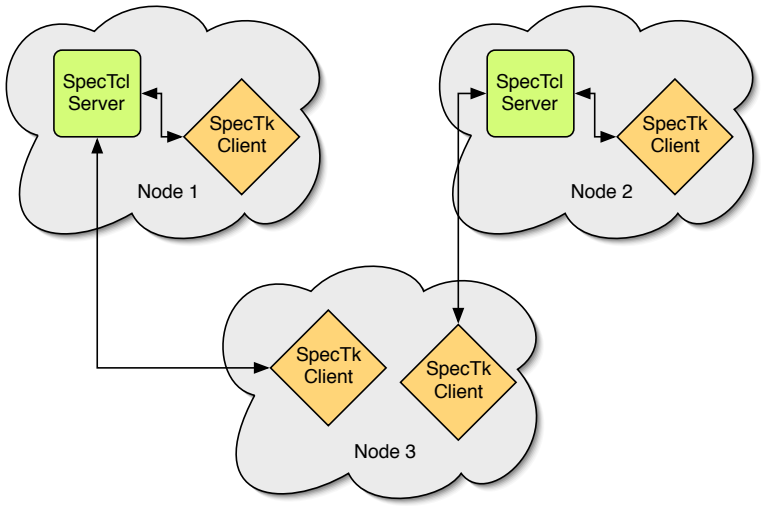

Figure 1: illustration of the client-server architecture used between the analysis program SpecTcl and the displayer SpecTk. The TCP/IP protocol used in Tcl/Tk allows connections across nodes. Note that the communication is double-sided, as the server can accept requests from any of its clients.

The last item in the requirement list can be nicely met using the client-server mechanism provided in Tcl/Tk. Using TCP/IP network connections between the NSCL SpecTcl data analysis program and one or more SpecTk displayer(s) is illustrated in figure 1. This architecture has several advantages:

- it isolates the displayer from the analysis program as a separate process, possibly on a different machine,
- many displayer clients can connect to the same analysis server, therefore the same data can be visualized by different users at different locations,
- a user may switch between different analysis servers without having to restart the displayer,
- in case the analysis server hangs, the data stored locally on the displayer can be saved,

• minimal modifications of the analysis program are needed.

The price to pay for this architecture is the time to transfer the histogram data between the server and its clients, which is in principle longer through a network connection than if the displayer were embedded in the analysis program. However, this can be efficiently optimized using binary format and compression, and doesn't limit the performance of the displayer with today's network performances.

In order to convert the SpecTcl analysis program into a server, a few modifications are necessary. Two additional Tcl commands to obtain and compress the histogram data are implemented in C++, for one- and two-dimensional data respectively. These commands automatically switch between absolute or indexed modes, depending on which results into the least amount of bytes to transmit. In the absolute mode, all bins of the histogram are transmitted, whereas only non-zero bins are taken into account in the indexed mode. The other modification consists of re-routing existing SpecTcl commands to make them communicate with the clients. That way, whenever one of these commands is issued in SpecTcl, it notifies all the clients which respond by updating the relevant information. This feedback mechanism makes the displayer(s) very responsive and is necessary in order for them to reflect the state of the analysis program at all times. The rename Tcl/Tk command provides a very elegant way of implementing this functionality. The Tcl/Tk server script added to the SpecTcl startup script simply renames the existing commands before redefining them. Each command then executes the original SpecTcl command and then notifies each of the clients with the same arguments. The server in SpecTcl is implemented as a separate safe interpreter which only responds to the relevant commands for security reasons.

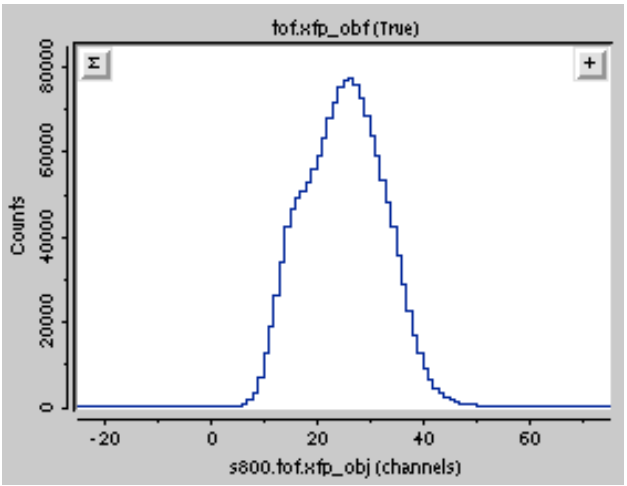

Figure 2: example of the representation of a one-dimensional histogram. The customary way of displaying the bins uses the step representation, but all other possibilities offered by the BLT package are available, including symbols with optional error bars.

3. Data representation

The BLT library provides objects called "vectors" which are one-dimensional arrays of real numbers. These vectors are intimately tied to the display widgets of BLT, which makes them the prime choice for storing the histogram data received from the server. In addition, several mathematical operations are available and can be efficiently used to perform statistical calculations on the data. The representation of one-dimensional his-

togram data in graph widgets is straightforward, thanks to the functionalities provided by the BLT library. The vector containing the data is simply displayed as an element of the graph, while the horizontal axis is set to the limits of the histogram (see figure 2). In case the number of bins is larger than the number of screen pixels, all bins are superimposed on the same pixel, which corresponds to the least loss of information.

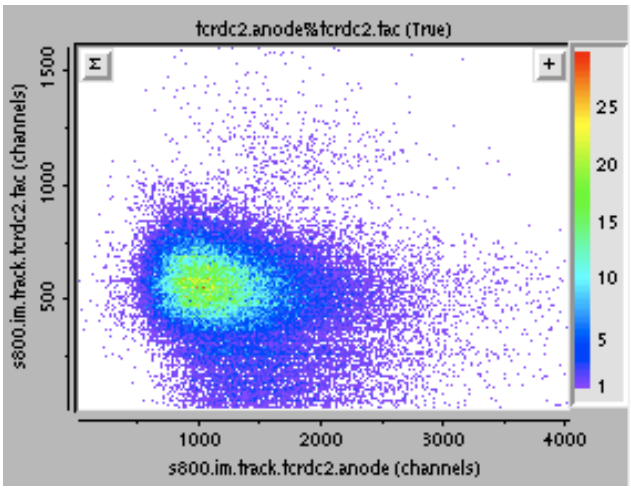

Figure 3: example of the representation of a two-dimensional histogram. Note the frequency color scale on the left.

Because in most cases the data in two-dimensional histograms is clustered, its storage is usually most memory-savvy using indexing rather than arrays. The other advantage of indexing is the ease it provides for performing statistical calculations. The most common way of representing two-dimensional histogram data uses color coding for the third dimension of frequency. Such a representation is not provided in the BLT package, and is implemented in C++ in SpecTk. The API provided for vectors in BLT, however, dramatically simplifies the task. The command Set2DImage implemented for that purpose takes the three BLT vectors containing the x and y indexes and frequencies corresponding to a two-dimensional histogram, and fills a Tcl photo image according to a chosen set of thresholds and colors. An example of the representation of a typical two-dimensional histogram is shown in figure 3. The Tcl photo image is simply superimposed on the BLT graph as an image marker while the axis limits are adjusted to the viewed area. Similarly to the one-dimensional case, only the bin with the maximum number of counts is displayed when fewer pixels than bins are available. This is consistent with the behavior of a surface plot where the smaller data is hidden under the surface.

4. Internal architecture

A simplified schematics of the internal architecture of SpecTk is shown on figure 4. Two different types of classes

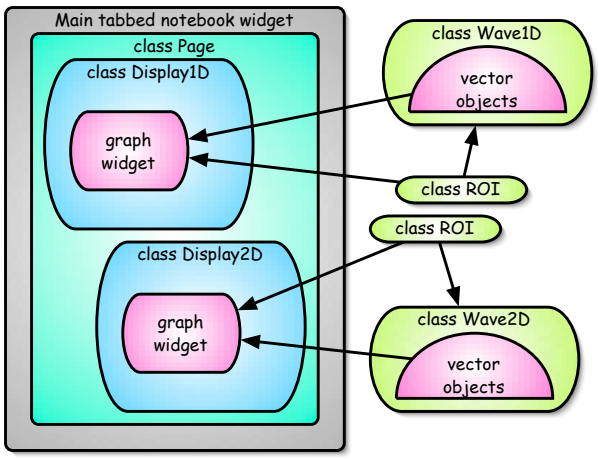

Figure 4: simplified diagram of SpecTk internal architecture. See text for details.

called Wave1D and Wave2D encapsulate one- and two-dimensional histogram data. Filtering objects such as region-of-interest (ROI) and gates are contained in ROI classes. The Wave classes contain BLT vectors to store the data, as well as all other information relevant to the histograms. The vectors are used directly as elements in the graph widgets for one-dimensional histograms, and through the Set2DImage command for two-dimensional histograms. The ROI objects are used by both the graph widgets to display filtering objects via markers, and the Wave classes to perform statistical calculations. The main display of SpecTk contains a tabnotebook widget in which pages can be defined. Each Page object in turn contains display panels in the form of Display1D and Display2D objects. The graph widgets are embedded in those two classes.

IV.Functionalities

In well designed applications, it is often desirable to provide the users with different ways of performing the same action. In the case of a displayer program, the typical example is the action of zooming in and out on a histogram or a selected group of histograms. Sometimes the user simply wants to select a region visually using a cursor, but other times precise limits of the region to observe have to be entered numerically. The displayer should be flexible enough to allow users to switch between modes easily. The following sections describe the functionalities of SpecTk where some of the redundancy just mentioned has been implemented.

1. Main display

The main display of the SpecTk application is shown on figure 5, where some of the features presented have been labeled with circled numbers. It should be noted that the overall color design of the application has been intentionally left in shades of gray to emphasize the data contained in the histogram, rather than drawing attention to the displayer itself. A detailed description of the features labeled by circled numbers follow.

(1) One cannot overstate the usefulness of tabbed pages widgets such as the tabnotebook provided in the BLT package. The convenience and real estate economy it provides on the screen is perfectly adapted to the display of pages of histograms. Not only can the user quickly switch between pages, but each page can also be detached as a separate window by clicking on the dashed line underneath the label. The configuration of each page is set and modified in one of the drawer panels (see section IV.2.a).

(2) Actions performed by the program can be applied to various combinations of displays. Selecting can be done in different modes, and the selected displays appear sunken and with a darker shade of gray. Note that any combination of histogram can be selected by pressing the Shift key while selecting.

(3) This cluster of buttons provides one of the means to inspect details of the displayed histograms. The bottom row respectively shrinks, sets to full scale or expands the viewed range of the selected histograms around the last previously expanded view. This is useful when quickly checking on the full histogram before returning to a specific region for instance. The middle row sets the frequency scale to either more of less number of counts, or back to Auto mode where the scale is automatically adjusted from the maximum number of counts. Finally, the top row allows users to switch between a linear or logarithmic scale for the frequency axis (the display on the right shows the same data as on the left, but with a logarithmic scale).

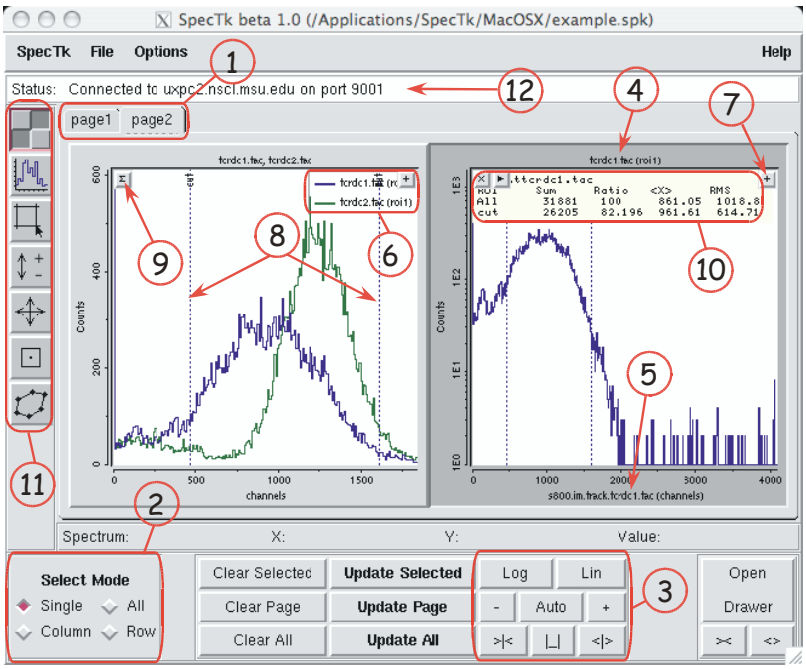

Figure 5: main display of SpecTk. See text for details. The circled labels point to the following features:

(1) page tabs
(2) selection mode in the page
(3) zooming and scale mode buttons
(4) histogram title and gating display
(5) parameter and units display
(6) legend if more than one histogram is displayed
(7) expand/shrink button
(8) region-of-interest (ROI) or gate display
(9) statistics button
(10) example of statistics display
(11) toolbar
(12) status display

(4) The histogram title identifies the display and shows the gating condition if one has been applied. This last piece of information is crucial because the user needs to know if any filtering conditions the histogram before an interpretation of the data can be sanely performed.

(5) The axis label show the parameter and its units. If more than one histogram is plotted on a given display, only the unit is shown. Only histograms of parameters using the same unit can be plotted simultaneously on a display, regardless of their number of bins.

(6) When more than one histogram is plotted on a display (this feature is only available for one-dimensional histograms so far), a legend is added to identify the traces. The gating condition of the individual histograms is shown in that legend.

(7) Each display of a page can be expanded to the full size of the page, and shrunk back to its original size using this little button on the top right corner. This feature is particularly easy to implement using the grid manager of Tcl/Tk.

(8) Any region-of-interest (ROI) or gate defined on a particular parameter shows automatically on the histograms of that parameter. ROIs are displayed with dashed lines, whereas gates are shown with solid lines. Notice the same example ROI shows on both displays of the same histogram.

(9) This button labeled with a Sigma ($\sum$) sign opens or closes a small window showing the results of statistical calculations.

(10) Example of statistical calculations. For each subset of data defined by any ROI or gate, as well as the whole data set in the histogram, the program calculates the summed number of counts, its ratio relative to the total, the average value of the parameter, and the root-mean-square deviation of the distribution. Those calculations are rendered extremely fast and easy to implement thanks to the vector objects provided in the BLT package and their associated set of mathematical operations.

(11) Toolbar. From top to bottom, the available tools are the following:

 Selection tool used to select histograms on which to perform actions. Various selection mode are available as already discussed in (2).

 Clicking on a display panel after selecting this tool opens a menu used to assign a particular histogram to the display. The way this menu is built depends on the choice of divider characters for the histogram names, as explained later in the Spectrum drawer tab section (section VI.2.b). Shift clicking allows the user to append histograms to a display. Only histograms based on parameters with the same unit as the ones already displayed are included in the menu. Shift right click is used to remove traces from the display, while right click aborts any of the previous operations. Note that while only this tool can be used to append or remove traces, the assignment of only one histogram to a display is easier done using the tree of the Spectrum drawer tab.

 This tool provides the visual method of inspecting details of histograms. A cross hair appears on the displays and is used to enter the limits of the viewed region. The first limit entered can be cancelled by using right click, and the full scale of the histogram can be restored using double click.

 The frequency scale of histograms can be visually set using this tool. For one-dimensional histogram, this sets the vertical scale, whereas for two-dimensional histograms the color scale is changed. Double click sets the frequency scale back to automatic scaling.

 This is yet another way of inspecting the data contained in histograms, but this time by scrolling a fixed range along the horizontal scale for one-dimensional histograms, and both horizontal and vertical scales for two-dimensional histograms.

 The inspect tool simply displays the contents of the histograms bin by bin, while the cursor jumps accordingly.

 The edit tool allows users to modify existing ROIs and gates displayed on top of histograms. For one-dimensional ROIs and gates, also called Slices, individual limits can be modified, but also both at the same time by holding the Shift key while dragging. For two-dimensional ROIs and gates, also called Contours, the whole polygon can be dragged when clicked inside, and individual points can also be relocated. This tool is a huge time-saver because ROIs and gates almost always need to be relocated as experimental conditions change. Whenever a gate is modified, the SpecTcl server is notified and updates the gate definition, which in turn dispatches it to all its displayer clients. This feedback mechanism ensures that all displayer clients reflect the current state of the gates used in the server. Also, statistical calculations are automatically updated whenever a ROI or gate is being modified.

(12) The status bar indicates whether SpecTk is connected to a server, and in case it is, which server on which port. If the server quits, SpecTk disconnects and notifies the user.

Other features of the main window include the menu bar, which contains the necessary commands for connecting and disconnecting from the server, saving or restoring SpecTk configuration files (the current configuration is indicated in the main window title bar), and some useful options such as the font type and size. The clearing and updating buttons are self-explanatory. The bottom right buttons operate on the drawer explained in more detail below.

2. Drawer tabs

The choice of a drawer for implementing the high level functionalities of the application was no doubt influenced by the Macintosh™ experience of the author. Drawers have the advantage of visually separating the functions it implements from the main window, while retaining the convenience of an easy access as compared to menus for instance. The drawer manipulation in SpecTk is operated by the Tcl/Tk grid manager.

Although it fulfills the required effect, one could consider implementing a more sophisticated drawer widget in future versions of Tcl/Tk. Because all functionalities cannot not possibly fit in the limited real estate of a drawer, once again the use of a tabbed notebook widget offered the most convenient solution. So far six tabbed drawer pages implement the following functions in a vertical arrangement.

a) Page definition (figure 6). The geometry - number of rows and columns - of display pages is easily managed in this drawer tab. Pages can be either created, modified or deleted, and their label edited. Note that page modification keeps the assignment of display panels untouched, thus users can easily add new displays to existing pages without having to re-assign all panels.

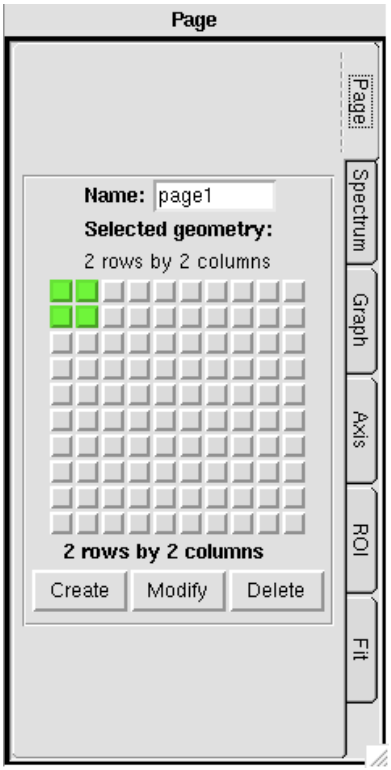

Figure 6: Page drawer tab.

b) Spectrum or histogram assignment (figure 7). This drawer tab provides an alternative way of assigning histograms to display panels than the menu-driven tool presented in the previous section. It uses the treeview widget provided in the BLT package to order the histograms according to their names. The branches of the tree are determined by a set of divider characters chosen by the user. Therefore, by adopting a naming scheme for the histograms and choosing the appropriate divider characters, the histograms can be ordered in a well structured tree. The example shown uses a dot (.) as divider character, while the histogram names are segmented using the same character. The resulting tree shows all the branches separated from the divider character, as well as the type of histogram or a colorful symbol ( ) for an internal name node. More than one divider character(s) can be used, and the menu used by the assignment tool ( ) is built on the same tree. The top part of the drawer shows information relevant to the highlighted histogram as the mouse hovers across the tree or the tree selection otherwise. The two buttons on the bottom can be used to assign the histogram to the selected display panel, the one marked "Selected++" automatically selects the following panel after the

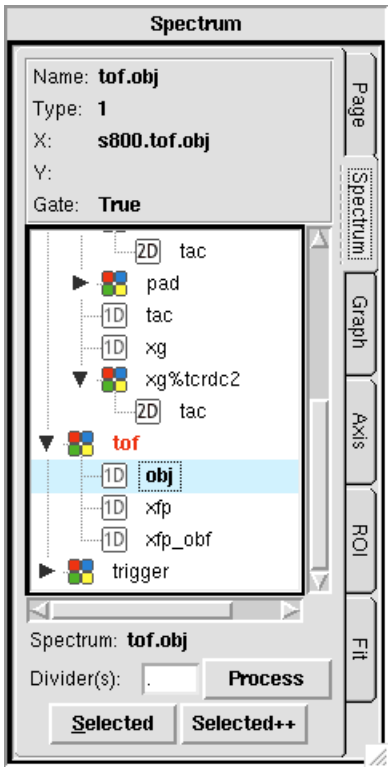

Figure 7: Spectrum drawer tab.

current assignment has been made. Alternatively, double clicking on the tree performs the same task. Using a tree structure increases the user efficiency tremendously when large numbers of histograms need to be handled, as often the case with multiple element detector systems nowadays used in nuclear and particle physics.

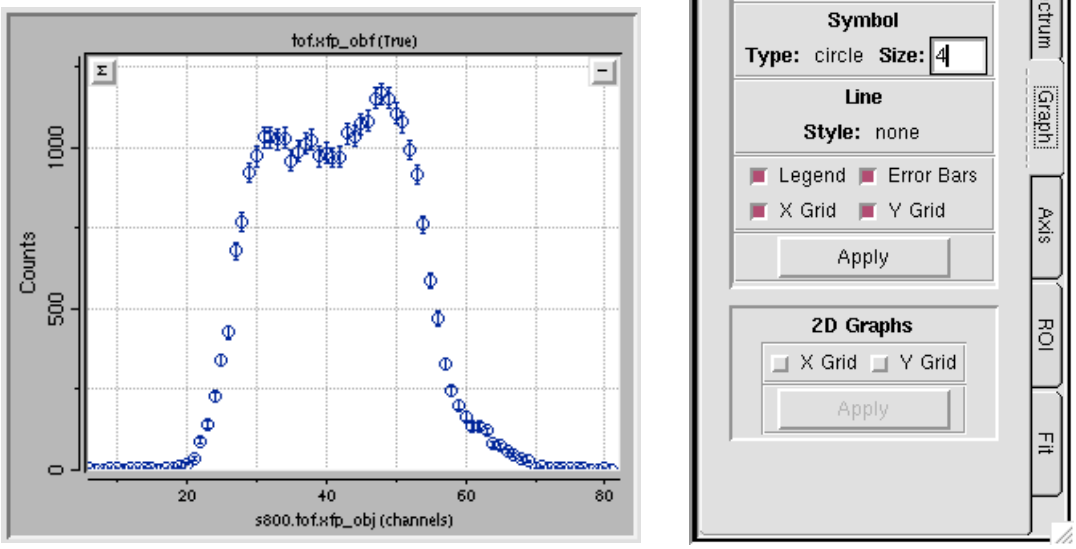

Figure 8: graph drawer tab and example of trace using symbols and error bars.

c) The next drawer tab (see figure 8) is basically an interface to the various graph element display options provided by the BLT package. They offer the possibility to change the representation of the histogram data using symbols, different types of lines between points, error bars and grids. An example corresponding to the setting shown is appended to the figure. The error bars are calculated from a normal statistical distribution, as the square root of the number of counts in each individual bin.

d) The last of the redundant ways of specifying limits for the viewed region of a histogram is implemented in the axis drawer tab (see figure 9). Here the user can type numerical values for the limits and apply them to selected histograms or to a whole page. Alternatively, existing limits of a single histogram can be loaded and propagated to others. This is particularly useful when comparing a set of histograms based on same or similar parameters

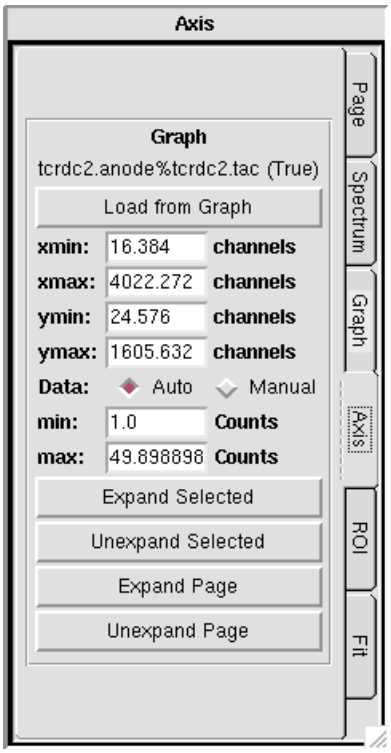

Figure 9: Axis drawer tab.

displayed in the same page. Both minimum and maximum limits of the frequency (or data) scale can also be manually adjusted, a feature often used in two-dimensional histograms to eliminate low-count background.

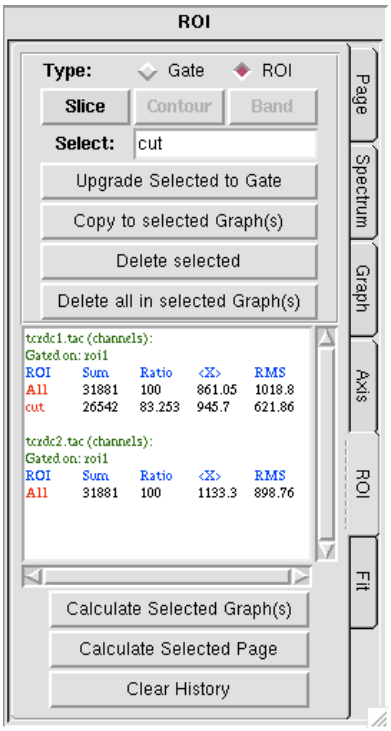

Figure 10: ROI drawer tab.

e) The region-of-interest (ROI) drawer tab manipulates filtering objects such as Slices and Contours (Bands are not supported at this point). The ROI category is internal to SpecTk, whereas the gate category is also used by SpecTcl to filter events. The choice of the gate category therefore triggers communication with the server, whenever a gate is defined, deleted or modified. The creation buttons initiate a modal sequence on the selected histogram, in which the user can either click directly on the display to enter the limits or use an entry box to type in a numerical value. At any stage of the entry sequence, the user can go backwards using the right mouse button. For two-dimensional histogram, the last point of a Contour polygon is entered with double click. After the entry sequence is complete, the program asks to either cancel or validate the entry with the name typed in the name entry box. Note that even though filtering objects are entered on histogram displays, they are actually bound to the parameter rather than the histogram itself. Other histograms based on the same parameter will automatically display the newly defined object. This feature is essential when comparing two histograms based on the same parameter but with different filtering conditions for instance. The ROI drawer tab also displays the statistical calculations performed on various histograms in a `text` Tcl widget. Since this widget support cut-and-paste operations, the user can easily export the results to other applications.

f) The last drawer tab implemented in this first version of SpecTk implements the data fitting functionality (see figure 11). The operation of fitting a data set with a given mathematical function is among the last steps in obtaining a measured "observable" from an experiment. The quality of the fit indicates how well the model represented by the function matches the data. SpecTk uses the classical least square fitting method, in which the quality of the fit is indicated by the value of the normalized X^2. A value of 1 indicates the best agreement between the fit and the data. For speed purposes, the fitting engine and functions are implemented in C++ rather than as a Tcl script. The number of fitting function is so far limited to four: Gaussian, Lorentzian, Exponential and Polynomial. Each has a linear function added to it, essential when fitting data sitting on

top of background. A formula for each function shows the definition of the parameters. The fitting region is defined by any ROI or gate Slice. The example shown in the figure

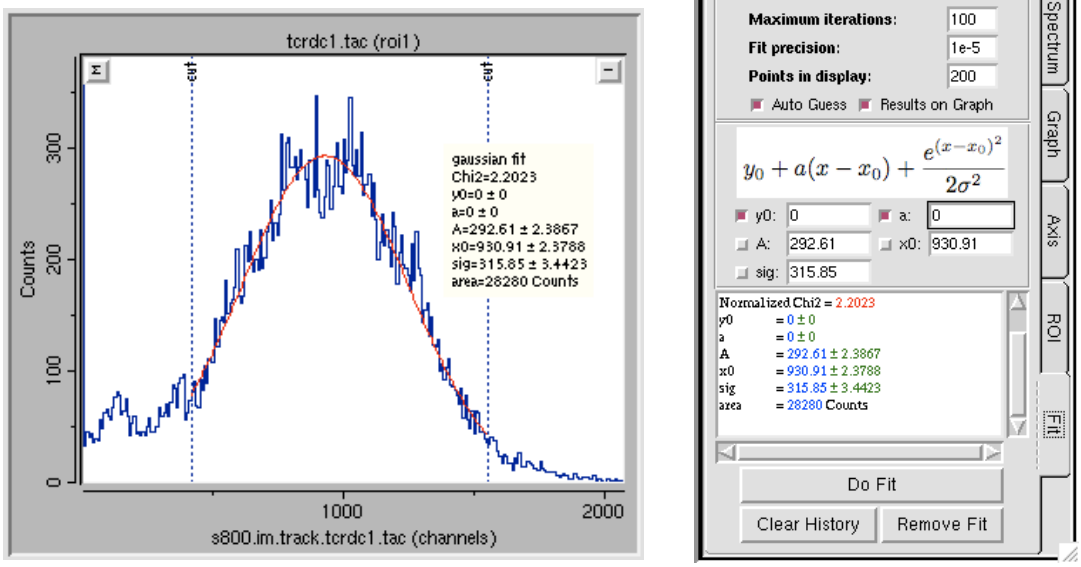

Figure 11: Fit drawer tab and example of a Gaussian fit.

is a Gaussian fit to the displayed data in the region defined by the ROI named "cut". Some of the fit parameters can be frozen to a chosen value by clicking on the check button next to them. The results from the fit appear in a `text` widget similar to the ROI results box, and the fitted function is displayed in red on top of the data in the histogram display. A small box containing the fit results can also appear on the display and be relocated to a convenient place.

3. Printing and fonts

Printing is usually a rather involved task for a programmer, if something better than a screenshot is to be provided. Not so with Tcl/Tk and the BLT library: the `graph` widget provides the necessary command to generate PostScript™ code for each of the display panels. The SpecTk printing dialog called from the

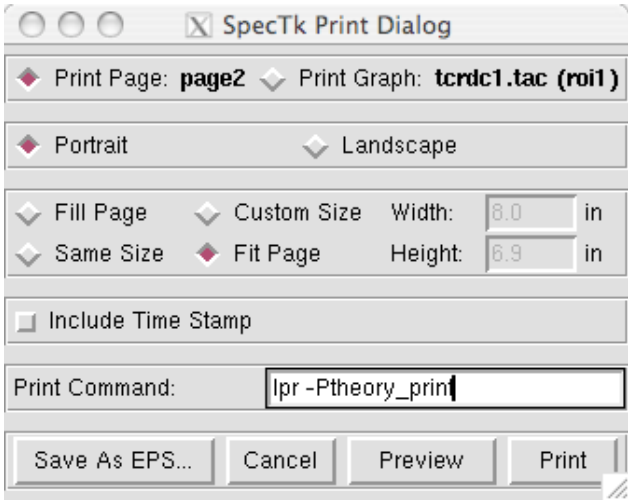

Figure 12: SpecTk print dialog. The user can choose to print a whole page of histograms or a single display. The size of the print can be easily adjusted. The printer name is specified in a standard Unix print command (not available under Windows™). A preview is provided and the resulting picture can be saved as a PostScript™ file. The fonts used in the final printout depend on the X Window installation.

"File" menu offers different options as shown on figure 12. The font rendering depends on the X Window installation of the machine running the program. The font dialog called from the "Options" menu allows the user to customize the various fonts in the program. Some trial and error is often necessary before the printed result is satisfactory due to variations in font implementation in printers as well. Figure 13 shows an example of a page printout saved as a PostScript™ file.

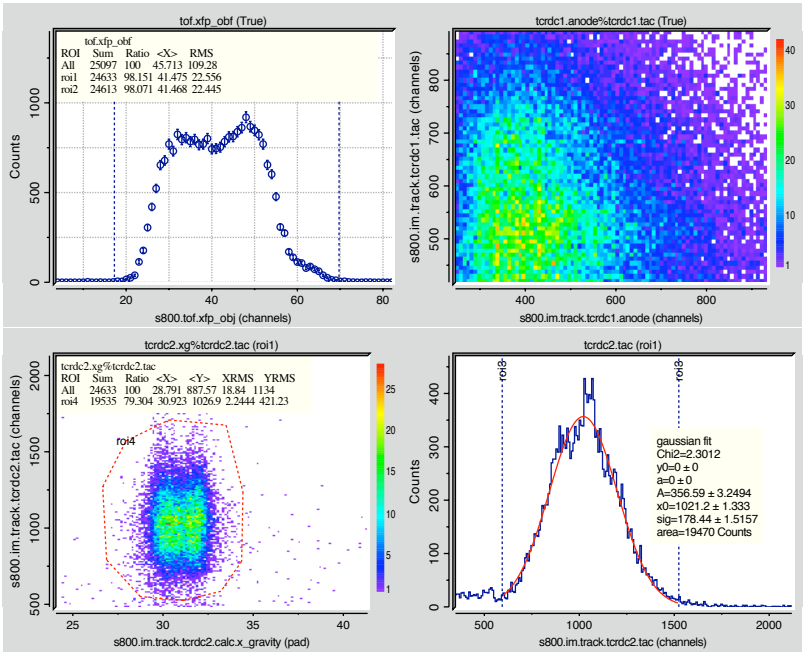

Figure 13: example of a page printout.

V. Future improvements

From a science usability point of view, improvements to such an application lie mostly in the realm of histogram manipulation. Once histograms have been accumulated in the data analysis SpecTcl program, with all necessary calculations and filtering conditions set up, they often need to be combined or manipulated before physical information can be extracted from them. A non-exhaustive list of often desired operations follows.

- Binning: the optimum binning of histogram data is obtained when the width of the bins is roughly an order of magnitude smaller than the features observed in the histogram. This way the maximum statistics in the bins is attained without compromising the significance of the data.
- Addition and subtraction: good examples are when histogram from many similar devices are added together to gain statistics, or background measurements need to be subtracted from the data. The problem of equal binning between the histograms before such operations are possible can be addressed with the previous binning operation.
- Multiplication by a constant: the frequency scale of histograms is invariably converted into physically meaningful unit such as a cross section for instance in case of nuclear and particle physics.

- Custom fitting: the mathematical functions used in physical science and nuclear physics in particular are often more complex than the simple four functions provided in SpecTk. The ideal tool would be to interactively enter a mathematical function as a Tcl script, which would then be used for the fit. Although in principle possible, this goal requires some thought in particular on how to manage the fit parameters between the Tcl and C++ worlds.

Other foreseen improvements include the possibility to save and restore histograms in various formats directly from SpecTk, and the addition of other two-dimensional histogram representations such as scatter plot.

The best future improvements are usually those requested by people who actually use the program, and see first-hand its shortcomings and what changes or additions would make it better. Since the user base of SpecTk is still rather limited so far, this kind of feedback hasn't happened very much yet, but is expected to grow in the near future.

VI.Conclusion

Originally developed on a Macintosh™ platform running the Unix-based OS X™ system, the program SpecTk has been successfully ported to Linux and Windows™ thanks to Tcl/Tk portability. This new displayer for the NSCL SpecTcl data analysis program is in its early phase of validation. A beta distribution is available at http://www.nscl.msu.edu/~bazin/SpecTk, where installation instructions are found in the install.htm file, as well as tar balls for the various supported platforms. It is worth mentioning again that this project wouldn't have even been started without the existence of the BLT package. From the knowledge of the author, this package is unique in its kind among the numerous expansions of the Tcl/Tk language, and is invaluable for all science-based applications using Tcl.