The following paper was originally published in the
Proceedings of the Fourth USENIX Tcl/Tk Workshop
Monterey, CA, July 10-13, 1996

High Performance Graphic Display With Tcl/Tk

Darren Spruce
European Synchrotron Radiation Facility
Grenoble, Franc

HIGH PERFORMANCE GRAPHIC DISPLAY WITH TCL/TK

Darren Spruce
European Synchrotron Radiation Facility
Grenoble, France
Email: spruce@esrf.fr

Abstract

This paper describes how Tcl/Tk was used for the development of a high speed updating graph display. Tcl is not reputed for its performance and where real time graphics is required a purpose built ‘C’ program is necessary to cope with the speed. However Tcl/Tk along with the BLT graph widget extension provide a useful set of tools for GUI building and graph displays. This application shows how performance can be built in to an interpreted environment so that the developer can benefit from the high level of functionality offered by Tcl/Tk and BLT and at the same time display a spectrum of 16000 points updating every second. Although the application is designed to display data from a Multi Channel Analyser (MCA), the technique can easily be adapted to display other types of 2 column data.

1 Introduction

The European Synchrotron Radiation Facility is a fundamental research institute situated in Grenoble, France. The ESRF ring is a third generation, the world’s most brilliant and intense source of X-rays to date, some years ahead of equivalent facilities in the United States and Japan. When complete the ESRF will have 50 beam lines dedicated to many fields of science.

My role in the programming group of the experiments division is to make existing programs easier to use by means of graphical interfaces, to develop software for instrument control and data analysis and to research into development techniques. At present my work is mostly involving Tcl/Tk and I am also supporting a growing Tcl/Tk user community.

2 Requirements for the MCA graph display

The Multi Channel Analyser is a device which counts electronic pulses and groups them into channels according to their height. It is typically attached to a detector and configured so that each channel corresponds to a different energy or time. Since the MCA only knows about channel numbers and counts, the channels need to be calibrated. The program needs to provide the possibility for the user to enter the calibration. The output is best be displayed as a graph of desired units (energy/time) versus counts, and this graph is changing in real time as the counts increase for each channel. Fast visual feedback during acquisition of data permits the user to make adjustments (for example during setup of the ADC).

Overall the application has to be easily adaptable to the varying requirements according to its particular use - some parameters are necessary for some users and unnecessary for others.

3 Choosing the Software

At the ESRF we have well established software for data acquisition. The problem was to redefine the user interface in a way to enable him to see and manipulate the data interactively as well as control the hardware.

What was needed was an independent application that could look at the data and present it in a user friendly and interactive way. It was necessary to look for high level tools to speed development and reduce the amount of work needed to complete the task because of the limited resources.

There were two suitable toolkits available, one was a commercial graph toolkit which could be used with a user interface builder, the other was Tcl/Tk with the BLT graph extension. The commercial graph widget required that a license fee should be paid for each piece of executable code that used it and since the ESRF is a public user facility where it is desirable to freely distribute software this is not very convenient. After comparing the functionality of both toolkits and seeing that they were similar I decided to first test the BLT toolkit for its performance.

The BLT graph widget proved to have lots of functionality but was not fast enough to cope with the amount of points and the refresh rate. So I resorted to using a ‘C’ routine which charged the BLT graph with the data. This was not difficult using the ‘standard’ methods to extend the Tcl commands and resulted in a much better performance.

The solution seemed to be to use the tools provided by Tcl/Tk and BLT to build a graphical interface and write some underlying ‘C’ code to handle the data. The transfer mechanism between the graphical interface program and the underlying software was implemented with shared memory. It is ideal for large amounts of data where synchronization is not too important and is compatible across the UNIX platforms used at the ESRF.

4 Design

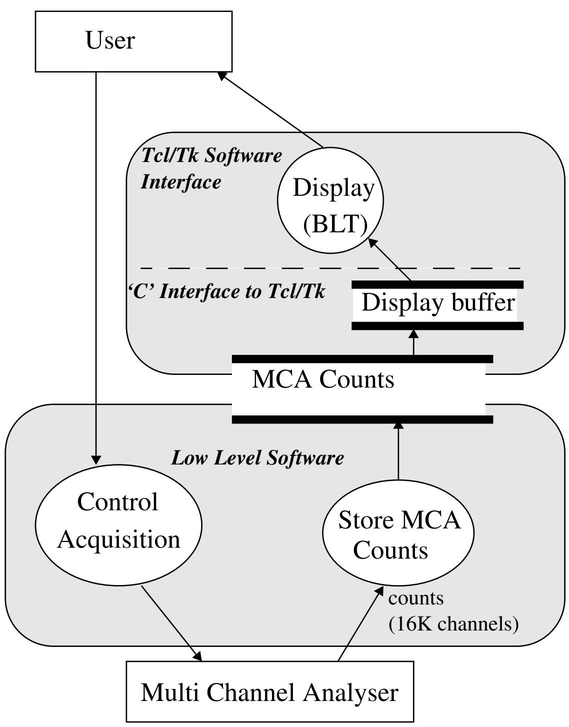

Fig 1. Data Flow Diagram showing User Interface to the MCA.

The data flow diagram above (Fig 1) shows the complete system to control the MCA and display the data. The object bridging the gap between the Tcl/Tk interface and the low level software is the shared memory area used to store/read the data. This is the only place where there is any dependency on the format of the information with the lower level software/ hardware.

The underlying software takes care of the details of talking to the equipment, the instrument control and the creation of the shared memory from which the MCA GUI collects its data. The data structure is the only constraint which ties the 2 pieces of software together. The structure used allows the update to be synchronised by the arrival of new data, and permits identification using a ‘magic number’.

Although the BLT graph could handle the refresh rate of the data it was inefficient in that it attempted to display all the points even if they were invisible due to the screen resolution. This loaded the X server heavily inhibiting other applications from running. However after a few days of further ‘C’ programming I improved the data formatting routine so that it reduced the raw data to the same equivalent number of points as the pixel width of the graph display. This changed dynamically as each call to update the graph included the current width of the graph. The modification successfully reduced the load on the X server.

5 Key Implementation Details

The following data structure defines the link between the underlying software and the display application.

```
struct shm_header {
 long magic; /* magic identifier */
 long type;  /* data type */
 long version; /* of shared memory */
 long rows; /* number of rows */
 long cols; /* number of columns */
 long utime; /* last updated time */
 char name[NAME_LENGTH];
 char spec_version[NAME_LENGTH];
 char pad[SHM_HEAD_SIZE (6*
sizeof(long) +2*NAME_LENGTH)];
 void    *data;
};
```

This structure originated from the underlying software and is used to implement the shared memory extended commands in the interpreter’s display.

The Tcl Interpreter is extended with a number of commands. Some are for analysis such as peaksearch and polynomial fitting, and others for calibrating the data. These are not covered in this paper.

The key Tcl extended commands are:

`get_shmid`

This commands reads the list of shared memory segments on the system and returns the id, name, version, number of rows and columns of the segment if it matches the pre-defined magic number. This permits the Tcl application to recognise and select between segments.

`read_shmid` `shared_id force_read`

Given the shared memory id and a flag this command checks to see if the memory has been updated by looking to see if the utime variable has changed.

`replot` `pathname min xmax width mode`

Takes the arguments, pathname of graph, minimum X value, maximum X value, the pixel width of the graph and the mode. This command gives big performance advantages over using Tcl to plot the data. It takes the currently selected buffer of raw data (there can be more than one) and reduces the data to the specified pixel width. It is also necessary to format and convert the type to make it suitable for BLT display call. The mode gives the possibility for the replot function to use a calibration function when converting the raw data.

The program execution starts by creating a window with a graph display and then initiates a poller which looks at the shared memory on the system. The poller code is written in Tcl with the extended command 'get_shmid' to read the shared memory. The name of the shared memory can be specified at runtime, so that if more than one application is running they can look at different areas of shared memory.

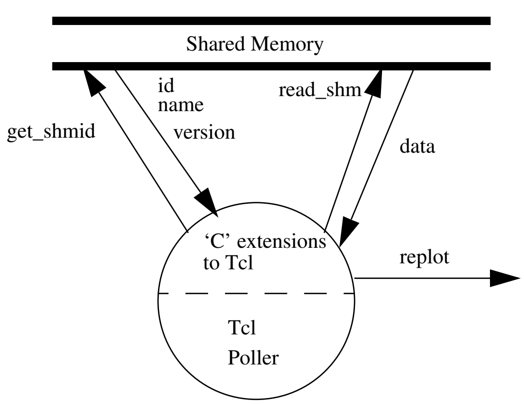

Fig 2. Poller interaction with shared memory

The poller runs slowly (once per second) unless it finds a matching identity. This gives the application more time to respond to user events when there is no shared memory to be updated. When an area of memory is found that matches the input arguments of the program the poller increases speed to approximately 10 times per second, now calling `read_shm` to read it if it has changed. If the contents have changed then they are read and the next 'test if changed' is scheduled in 0.3 seconds. This means the memory will be read at a maximum rate of 3 times per second if it is always updating and the delay before an update is about 0.1 seconds. The data is read into an intermediate buffer in raw form before it is formatted for the BLT graph.

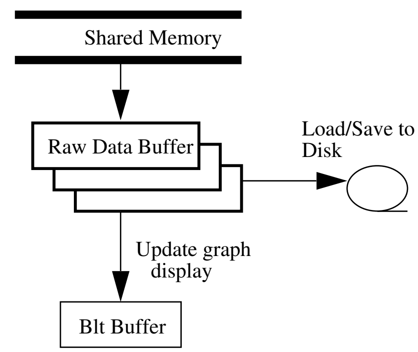

Fig 3. Buffers used by the application

There are 2 buffers in the program, one for the raw data and the other for the graph display. Each time a change is made which requires a re-plot of the display (e.g. window resize, zoom in/out or scroll) the **`replot`** command is called which supplies the graph width, range and calibration mode to the plot routine. This routine formats the relevant area of raw data and then copies it into the graph widget buffer. This is necessary to be able to save or load the raw data since not all of it is contained in the Blt buffer. It also allows multiple plots in the same window.

Every time the graph display is updated either by resizing the window, zooming or scrolling the ratio of pixel points against data points over the zoomed region is re-calculated and the corresponding number of points read from the memory buffer to the blt buffer.

6 Program Functionality

This section describes the various functions of the program. It is included to give a flavour of the high level

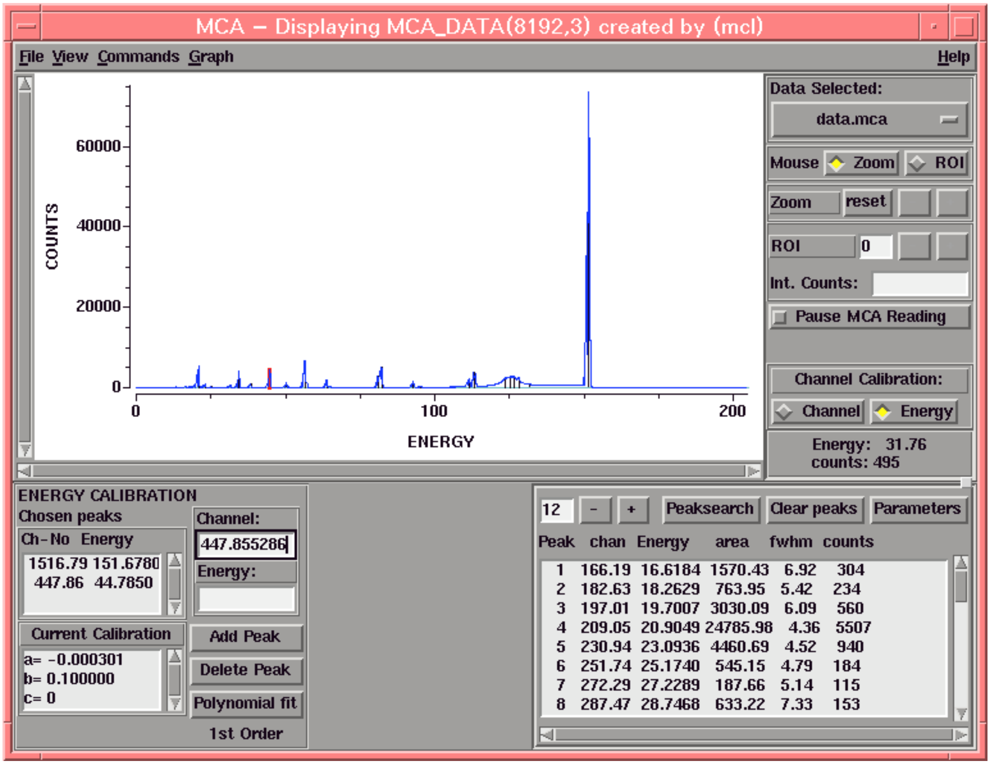

Fig 4. Overall appearance of the Window

of functionality achieved by using the Tcl/Tk/Blt/Tix toolkits.

6.1 Window creation

The main window was originally built using only Tcl/Tk and BLT. It was later modified to make use of the advanced features of the Tix widget toolkit such as the file selection box and paned windows. The other main benefit of using Tix was the very neat default colours and appearance.

6.2 Update Graph Poller

See ‘Key Implementation Details’ for information on the poller.

6.3 Graph control

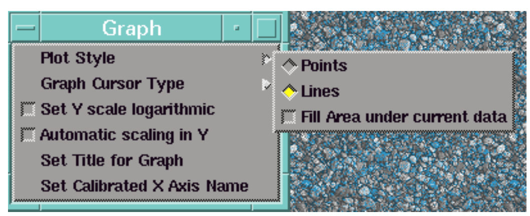

Fig 5. Graph Style Menu

Following an action by the user the graph often needs updating. The Tcl code handles the parameters of re-plotting and then calls a ‘C’ routine to update the plot. The display styles of the plot are also controlled with Tcl. Originally I used a demonstration program supplied with BLT to control all its resources and this provided a good way to experiment with the graph appearance. Finally it was necessary to limit this to a more manageable set of display options.

6.4 Peak Search

The user can start a peak search. This will look for peaks in the currently selected plot. A ‘C’ routine looks for peaks, and the positions are returned to Tcl. The Tcl code handles user selection and displaying of the peaks and also a way to enter them into the calibration.

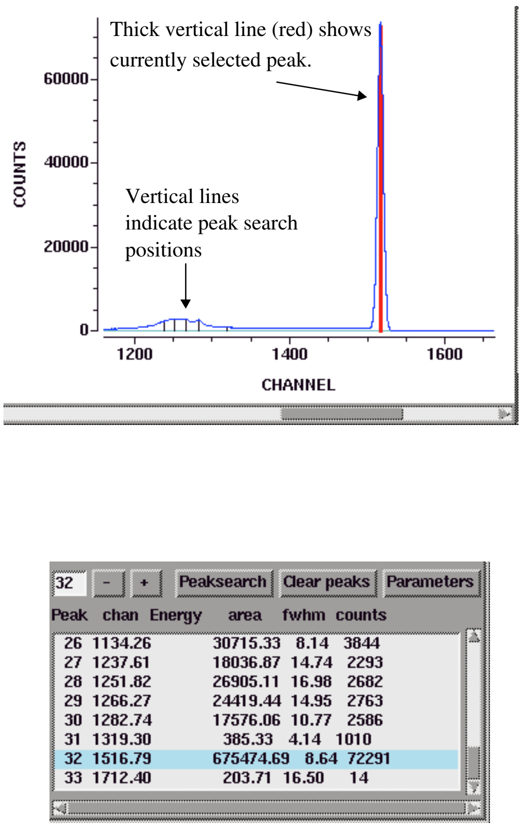

Fig 6. Peak Search facilities

6.5 Regions of Interest

It is possible to define areas of the graph marked by vertical lines defined by the user with the mouse. When selected (shaded in grey) any peak search is only done over the range of the shared region. The integrated counts over that region are also calculated and displayed. The regions of interest are definable by dragging with the mouse or by entering the boundaries using entry boxes.

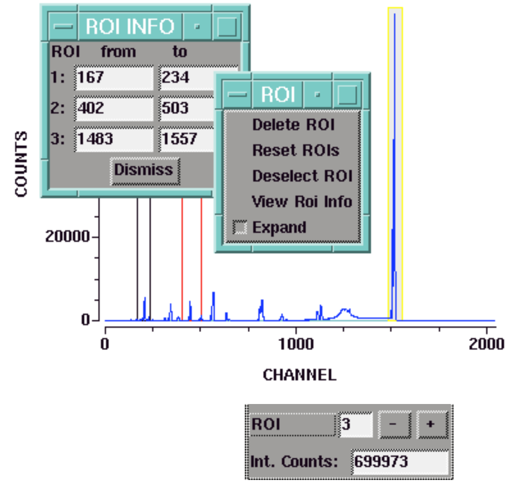

Fig 7. Selection of windows for ROI selection.

6.6 Zoom Facilities

Zooming can be performed either with the mouse by clicking and dragging over a region or by entering the desired boundaries of the zoom region. Each zoom is stored in a list which can be reset at any time. There is also an expand function for regions of interest where a zoom is performed to the limits of the currently selected region of interest. A zoom can be performed while the display is changing and so it is possible to choose whether the graph continues to do automatic scaling in the y axis or to freeze the zoomed region.

6.7 Scrolling

Each time the user zooms in it is still possible to scroll over the whole range of the data. The graph updating is handled by ‘C’ code because the BLT graph does not contain all of the data. When scrolling the calculations for the scroll bar sizes and the scroll step size is done with tcl and the ‘C’ function to update the graph display is called.

7 Performance

The running program consumes between 7% and 15% of the CPU of an HP9000/715 machine along with a similar load on the X server. This leaves enough resources of the machine for the underlying application which reads the MCA data and for other beam line applications. One interesting feature is that the CPU usage of the program and the X server is proportional to the window size as the amount of work to be done to refresh the graph is related to on the number of pixel points to be plotted. If the user is less interested in the display he can reduce the window size therefore reducing the CPU load. Only when he becomes more interested in the display does he enlarge the window and increase the CPU load. The screen refresh rate is at about 2 or 3 times per second depending on the CPU load and if you are using an X terminal, the amount of network traffic.

8 Conclusion

Tcl/Tk, BLT and a later addition the TIX extension proved to be robust software toolkits which were easy to build on, providing a high amount of functionality coupled with easy extendability. This example has showed how the best of both worlds (performance and high level programming) can be achieved by using an interpreted language.

9 References

[1] John K. Ousterhout: An Introduction to Tcl and Tk Computer Science Division. Department of Electrical Engineering and Computer Sciences. University of California, Berkeley. Draft for Addison Wesley 1993

[2] George Howlett: Blt 1.7, Blt1.8.
gah@grenache

[3] Ioi Lam, Tix 4.0b8 1995
ioi@CS.Cornell.EDU

[4] Brent B. Welch:
'Practical Programming in Tcl and Tk',
Prentice Hall PTR 1995