

A GRAPHICAL USER INTERFACE FOR RELAX3D

F.W. Jones, TRIUMF, 4004 Wesbrook Mall, Vancouver, B.C., Canada V6T 2A3

Abstract

The Laplace/Poisson solver Relax3d has been used extensively in cyclotron central region design and other accelerator and beam physics applications. It is typically run in an interactive mode where the user types in commands and parameters to initiate and control the solution process and to view or output the results. This paper describes a graphical user interface (GUI) that eliminates most of this typing and makes for more efficient user interaction. The use of a unique package called Expect (an extension to Tcl/Tk) allows the interface to be implemented as an independent front-end process that communicates with the running Relax3d program, thus requiring minimal modifications to Relax3d itself. Since Expect can control multiple processes, and since Relax3d results are often sent to some subsequent program for visualization, particle tracking, etc., there are interesting opportunities to integrate these post-processing tasks into the same GUI.

1 INTRODUCTION

In the field of Accelerator Physics there is a large legacy of Fortran codes, many still widely used, that were developed in the text-based environment of previous computing platforms. Many of these codes could benefit from the addition of graphical user interfaces. For batch programs that have many input parameters, or for interactive programs that have many different commands and command options, these interfaces can be tremendous labour-saving devices. Moreover, a well-designed GUI represents a program's parameters and functions and in a direct visual way that allows the user to largely dispense with reading voluminous documentation and memorizing mnemonics and keywords.

Relax3d[1], as one such Fortran code, is an ideal candidate for a point-and-click interface. It has a repertoire of around 30 interactive commands, many with parameters (up to 6). Its high level of interactivity has made Relax3d a popular and relatively easy-to-use program, but the process of working through the entire design, set-up, and solution process for a given problem, with initial trials, production runs, and checking and visualization of results, can involve a lot of typing and a certain sense of tedium.

2 INTERFACE OPTIONS

The X-based OSF/Motif toolkit, available on many computing platforms, was considered as a means for implementing a GUI. This raised some general concerns: (1) Motif and other toolkits impose an asynchronous event-driven protocol: applications must have an event loop as their core structure and must respond to any input event (e.g. a mouse

click) that can occur. This represents a "clash of architectures" with many Fortran programs, which expect particular items of input at particular times and may be unresponsive to input at other times. Extensive modifications may be required to implement the GUI. (2) The interface code must, or should, be written in C or C++ (for various good reasons, GUI programming is rarely done in Fortran). This escalates the application to a mixed-language environment, where argument-passing and name-space protocols have to be followed carefully and are not always portable between platforms.

These issues can be dealt with, but the fatal problem for Relax3d is the need for the user to rebuild the Relax3d executable on-the-fly, to incorporate the user-written BND subroutine that defines the boundary conditions for a given problem. It would be unreasonable to expect that the Motif library, or some other toolkit library, would be available on the platform where Relax3d is being used. Moreover, linking in such a large and complex library could slow down the problem-solving process which often involves a number of trials with different BND routines.

A possible solution to this is to implement the Motif GUI as a separate program written in C, which would run as a distinct process (front-end) and would communicate with Relax3d (back-end) via UNIX pipes. This approach also avoids the mixed-language concerns and isolates the event-handling architecture in the front-end.

3 PROTOTYPES

A prototype of this loosely-coupled type of interface was written, with the help of the Motif GUI construction tool XDesigner, and tested on various UNIX platforms. A crucial requirement for this system was the need to interrupt long-running Relax3d computations, in order for the front-end to gain control and possibly make changes or corrections. Relax3d itself already has the mechanism for this, a handler for the INTERRUPT signal, usually sent by pressing CTRL-C on the keyboard.

It was no problem to have the front-end send the same interrupt signal, but during testing it was found that under certain conditions the signal would consistently fail to get delivered to the Relax3d process, rendering it uninterruptible and leaving the user to either wait or kill the process and start again.

To try to isolate this fault, and also to investigate alternative toolkits, another interface with almost identical features was written using the popular Tcl/Tk scripting/GUI language. This interface was easily and rapidly prototyped, even without the help of an XDesigner-style GUI builder, because of Tcl's simple syntax and its interpretive shell

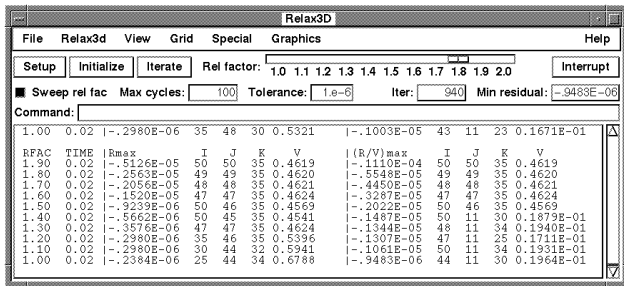


Figure 1: Main window

which allows the Tk widgets to be deployed and configured on-the-fly. This prototype worked well but did not solve the signal delivery problem, which appeared not to be an interface problem but rather a consequence of Relax3d reading input from a pipe rather than a terminal.

4 EXPECT

During the prototype development there arose another avenue of pursuit: a Tcl/Tk extension package called Expect[2]. This package was originally developed as a tool for communicating with pre-existing interactive programs, such as UNIX system utilities, in order to automate frequently-performed tasks.

As an extension, Expect provides all of the Tcl/Tk functionality together with a set of added commands that provide a flexible and programmable facility for launching and communicating with external programs. With the Tk widgets ready to hand, it proved to be an excellent environment for constructing a GUI for Relax3d. Best of all, the initial tests showed that Expect solved the interrupt signal problem. Expect does not use pipes, rather it creates UNIX *pty* (pseudo-terminal) devices to communicate with spawned processes, which thus behave just as if they are reading from and writing to a real terminal.

The other capabilities provided by Expect were the icing on the cake. Notable among them are: (1) the ability to spawn and communicate with multiple independent processes, (2) a programmable event-loop type of input processing to pattern-match against the output from spawned processes and take appropriate actions depending on what is received, (3) a timeout mechanism to keep the front-end from hanging while waiting for a spawned process to produce output, and (4) the ability to connect the user's terminal window directly to a spawned process. The importance of this last feature will be enlarged upon later.

Although most of the example Expect applications in the literature are fairly small and task-specific, further work on the Relax3d prototype showed that the system could be scaled up very nicely into a complete, multi-window interface.

5 INTERFACE OVERVIEW

The basic program functions, each corresponding to a Relax3d command, are invoked from the Main Window (see Figure 1). For efficiency, simple pushbuttons are provided

for the most heavily-used commands: Setup to specify the problem type and grid parameters, Initialize to set the grid to initial values, Iterate to start the iteration process, and Interrupt to prematurely stop it in the manner discussed above. The relaxation factor, often an item to be experimented with, is set by a slider, and editable text input fields Max cycles and Tolerance are used to set stopping criteria based on number of iterations or maximum normalized residual. Using an Expect directive, the line-by-line output from Relax3d's iteration loop is captured and displayed in a scrolling text area, providing a complete record of the run. In addition, this output is scanned to extract the residual and display a running minimum in the Min residual text field.

The remaining program functions and settings are accessed via popdown menus on the menu bar. Some of these functions could be implemented directly in a few lines of Tcl/Expect code, whereas others were written as separate Tcl procedures that bring up independent dialog windows. These procedures all follow a common pattern: (1) If the window has not been created then create it, otherwise pop it up if it is iconized or off-screen. (2) When the user clicks one of the "action" buttons in the window, build the appropriate command string and send it to Relax3d's input stream. (3) Pattern-match the output from Relax3d and take appropriate actions such as sending more input or displaying error messages.

The following example, edited for brevity, shows part of this sequence for the Contour Plot procedure (Relax3d's PLOT command). The complete plot window is shown in Figure 2.

```
proc plot {} {
    ...
    # If plot window is already made, map it and return
    if [winfo exists .p] {
        wm deiconify .p; raise .p
        return
    }
    # Construct the plot window
    toplevel .p
    wm title .p "Contour Plot"
    ...
    checkbutton .p.yes -text "Label contours" \
        -variable clabel
    # Text fields for label spacing and plot magn:
    entry .p.spacing -textvariable spacing -width 5
    entry .p.mag -textvariable mag -width 4
    ...
    # Button to send Plot command and options to Relax3d:
    button .p.action.plot -text PLOT -command {
        set dolabel ""
        if {$clabel == 0} { set dolabel "-" }
        exp_send "PLOT $dolabel$spacing $mag \
            $xorig $yorig $stype\r"
    }
    # Pattern-match Relax3d o/p & diagnose errors:
    expect {
        "Option number" { }
        "*Relax3D > " { errmsg $expect_out(buffer)
            return }
    }
    ...
}
```

Tcl's string-oriented syntax makes it easy to assemble the appropriate command, with parameters taken directly from variables tied to the Tk widgets, and ship it to Relax3d with the `exp_send` directive. Then the `expect` loop checks for possible outcomes: a prompt for more input

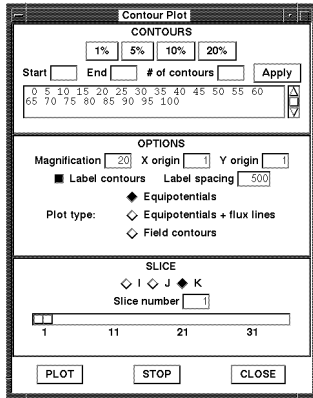


Figure 2: Contour plot window

indicates that the command was accepted, whereas some text followed by the `Relax3d>` prompt indicates an invalid command parameter and the captured error text is displayed in a message box.

Some other interesting features of this interface are: (1) A `Rebuild` button that shuts down the current `Relax3d` executable, builds a new one from the selected BND routine, starts it up, and connects the GUI to it. (2) An option for redrawing a contour plot on-the-fly as iterations proceed. (3) An `Interact` button that connects the user's terminal window to the spawned `Relax3d` process, allowing the user to assume "manual control" of `Relax3d` in case of any doubt as to what is going on. This utilizes Expect's `interact` command, a very powerful feature indeed for both users and developers, the more so because while the terminal is connected to `Relax3d`, the GUI remains connected and fully operational as well (the best of both worlds)!

6 BACK-END MODIFICATIONS

`Relax3d` itself has required some modifications to support a GUI front-end, but they are relatively few and straight forward. Some new "private" commands were added to allow the interface to inquire on the state of the program and the values of various set-up parameters after a problem set-up has been performed. Some procedures were modified to produce more diagnostic output that allows the interface to monitor their progress, and some "hand-shaking" was implemented where large amounts of data had to be transferred between the GUI and the program. None of the modifications interfere with the use of `Relax3d` in the normal terminal mode without the front-end.

7 VISUALIZATION

In the Expect environment, it was a natural progression from interfacing to the built-in graphics to interfacing to more powerful external facilities for producing graphics. Programs like `GnuPlot`, `Mathematica`, and `MATLAB` were investigated for this purpose and `MATLAB` was chosen for its excellent publication-quality 2d and 3d graphics facilities and its ease of use. It proved to be almost trivial

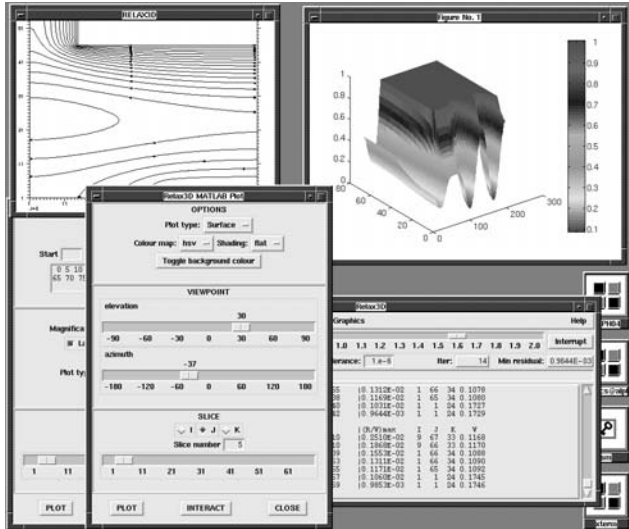


Figure 3: Relax3d session with MATLAB interface

to write the additional `Tcl/Expect` code to launch `MATLAB` from the `Relax3d` interface, modify `Relax3d` to write out slice data in a form digestible by `MATLAB`, and then build a prototype "MATLAB Plot" dialog window offering various 2d (contour, pseudocolour) and 3d (waterfall, mesh, surface, slice) plot types, with menus and sliders for colourmaps, viewing angles, etc. Once again, Expect's `interact` command is invaluable since it allows the user access to all of `MATLAB`'s commands without disabling the GUI operation.

8 CONCLUSION

The software technology used in the `Relax3d` GUI is sufficiently general and flexible to have a wide range of applicability to Accelerator Physics computing: lattice and beam-line design codes, tracking and raytracing codes, and simulation codes are some obvious examples where efficient interactive run preparation and visualization facilities are desirable. In general, the Expect package can be thought of as a way to interlink computer programs to create problem-solving environments. Together with `Tcl/Tk`, it can provide a lot of functionality for a relatively modest investment of programming time.

Further development of the `Relax3d` GUI will likely concentrate on electric field plotting methods, volumetric visualization, and the development of a spreadsheet-style viewing facility for navigating through the grid to check boundary and solution values. Those wishing to obtain `Relax3d` and/or the first released version of the GUI can connect their Web browsers to <http://www.triumf.ca/~compserv/relax3d.html> for up-to-date information.

9 REFERENCES

- [1] H. Houtman, F.W. Jones, and C.J. Kost, "Solution of Laplace and Poisson Equations by RELAX3D," *Computers in Physics*, July/August 1994.
- [2] D. Libes, *Exploring Expect*, O'Reilly & Associates, 1995.