Stanford Exploration Project, Report 67, January 13, 1998, pages 1–151

Stanford Exploration Project, Report 67, January 13, 1998, pages 1–151

Progress towards the interactive book

Martin Karrenbach and Dave Nichols [1]

keywords: *interactive, algorithm*

ABSTRACT

We have been able to use recent advances in workstation technology and software to make progress towards Jon's goal of an "interactive book". By using the "xtex" previewer we can preview documents written in the typesetting language TEX. Furthermore by using special TEX commands we can add "hot spots" to the previewed document, when the user clicks on these areas a sequence of commands is executed on the workstation. This ability makes the document interactive, the text can describe an idea and then clicking on a button will start a program that shows the idea in action. Clicking on the button can also regenerate a figure within the text, either the same figure or a version made with new parameters.

INTRODUCTION

Most scientific workstations sold today use some variant of the Unix™ operating system and are capable of supporting the X window system. Almost any workstation meeting these requirements can run a program to preview typeset documents in "dvi" format. The documents in this format are usually produced by text processors that take either TEX or troff files as input. At SEP we produce all our reports using TEX. A more recent advance is the possibility of viewing figures as well as text with the previewer. Since PostScript™ is becoming a de-facto standard for exchanging figures some previewers are now able to display PostScript figures within the text. The ability to preview a whole document is very pleasing. Instead of printing 10 trial versions of a paper we can check them on the screen and make updates as necessary. Not only does this save trips to the printer it also saves time, money and trees.

A previewer like this is only a one way device, it displays the document in a static form, the user can learn no more form the previewed document than they could from a paper copy. The major advance that we have made recently is to allow interaction between the user and the document. As a result of our communications with the author of the "xtex" previewer (Dirk Grunwald at Colorado Univ.) he produced a version of the program that can react to mouse clicks on the previewed page and invoke *any* Unix command.

[1]**email:** not available

This flexibility opens up an enormous field of possibilities. The temptation to fill a document with buttons that spin off arbitrary commands is great. However in order to prevent the user from becoming overwhelmed by interaction we must define a consistent interface between button presses and system commands. In this example we use the utility program *cake* as an interface between the xtex button presses and the Unix commands. The dependency analysis performed by cake ensures that only the minimum amount of work necessary to complete a command is done.

HOW TO DEFINE INTERACTION

Authors have the ability to place the command `\syscall{text}{action}` at any point in the TeX document. This command takes two arguments; the first one `text` is the text that appears in the document with a box around it. Thus the command `\syscall{text}{action}` will appear as . Hitting the middle mouse button within such a box, will issue a system call with the unix command `action`. This TeX system call is how the author and reader interact.

For convenience there are plot commands for figures, which automatically connect the figure filename with an action stored in a cakefile. This cakefile is in the same directory as the TeX document. Thus typing `\activeplot{name}{height}{caption}` will create an active figure. This command includes a PostScript file, given in file `name.ps`, specifies the `height` of the figure and puts some `caption` on the bottom. The figure caption will include a button labeled: `name`. When the button is pressed the system call: `cake name` will be executed.

HETEROGENEOUS DOCUMENTS

The step of forming one interactive document out of a collection of several interactive documents is an important issue. Usually a book is divided in several chapters, or a report consists of a couple of individual articles. Such a heterogeneous collection of documents can each reside in a different location (directory). Ideally this directory contains the whole environment (text, figures, programs and command scripts) for reproducing and defining interactions for the entire interactive document. Such an object-oriented philosophy (OOP), as described by Jon Claerbout in this report, requires that while including objects, such as articles or chapters, their interactivity is maintained. This requirement is not trivial, since an interactive document object usually consists of a directory or structured subdirectory tree.

We solved this problem in LaTeX by creating a new `\include{path}{file}` command, which has two arguments. The first is the directory path name, indicating the location of the object-oriented document; the second is the name of the file that contains the actual text. This new command differs from the regular one in that it automatically specifies the current working directory to be that of the included document. In practice, this means when then reader views a heterogeneous document, the current working directory associated with the interactivity changes from chapter to chapter or from article to article. The reader himself does not notice the change of environment, as he presses buttons to start up some action.

It is not readily apparent what the best granularity of a heterogeneous document will be. As a starting point we have chosen an object-oriented document to be a self-contained article or book chapter. For other applications a different granularity may be more suitable.

EXAMPLE DOCUMENT

The following example is a hardcopy printout of an interactive document. The figures following it show examples of the interactive document in use. After the figures the literal TeX file is given, followed by the `cakefile` used for this document. Knowing that a reader really should read this shortnote on the screen in an interactive fashion, we hope that even a plain paper copy gets readers interested in reading documents interactively.

CONCLUSIONS

The mechanism provided by `xtex` for generating actions is response to user input is extremely general. It is so general that we need to carefully impose our own structure on the actions. This is necessary to ensure that the users and the document authors perceive a consistent interface. In this example we have used the program `cake` to bind the users actions to processing steps. The reasons for this choice are discussed in the following articles in this report.

REFERENCES

Claerbout, J., 1990, Active documents and reproducible results: SEP–**67**.

Claerbout, J., and Nichols, D., 1990, Why active documents need cake: SEP–**67**.

Hardcopy of the Example Document

Exercise: (ω,x,z)-Wavefield Extrapolation

This document is an actual exercise taken out of Jon Claerbout's book *Imaging the Earth's Interior*. This exercise performs a wavefield extrapolation in (ω,x,z)-space. At the top boundary data are given as if recorded from a spherically diverging wave coming from within the medium. The goal of this exercise is to show that the (ω,x,z)-space can be used for downward extrapolation of recorded data. The following is a listing of the existing program written in ratfor. It is directly included from the directory in which this document is stored.

```
# Wavefield extrapolation in (omega,x,z)-space
#
# Keywords: extrapolation movie class

#implicit none
complex cd(96),ce(96),cf(96),q(96),aa,a,b,c,cshift
real p(96,96,12),phase,pi2,dx,dz,v,z0,x0,dt,dw,lambda,w,wov,x
integer ix,nx,iz,nz,iw,nw,it,nt,outfd,output

nt=12; nx=96; nz=96; dx=1.; dz=1.; pi2=2.*3.141592
v=1; lambda=nz*dz/4.; dw=v*pi2/lambda; dt=pi2/(nt*dw); nw=2;

call getch('nw','i',nw); call putch('nw','i',nw);
call putch('n1','i',nz); call putch('n2','i',nx); call putch('n3','i',nt);
# For plotting:
call putch('ras1','i',12); call putch('ras2','i',12); call putch('gpow','i',1);

outfd = output()
call hclose()

do iw = 1,nw {                                  # superimpose nw frequencies
        w =  iw*dw; wov = w/v                   # frequency / velocity
        x0 = nx*dx/3.; z0 = nz*dz/3
        do ix = 1,nx {                          # initial conditions for a
                x = ix*dx-x0;                   # collapsing spherical wave
                phase = -wov*sqrt(z0**2+x**2)
                q(ix) = cexp(cmplx(0.,phase))
                }
        aa = (0.,1.)*dz/(4.*dx**2*wov)          # tridiagonal matrix coef
        a = -aa;      b = 1.+2.*aa;      c = -aa
        do iz = 1,nz {                          # extrapolation in depth
                do ix = 2,nx-1                  # diffraction term
                        cd(ix) = aa*q(ix+1) + (1.-2.*aa)*q(ix) + aa*q(ix-1)
                cd(1) = 0.;      cd(nx) = 0.
                call ctris(nx,-a,a,b,c,-c,cd,q,ce,cf)
                        # Solves complex tridiagonal equations
                cshift = cexp(cmplx(0.,wov*dz))
                do ix = 1,nx                    # shifting term
                        q(ix) = q(ix) * cshift
                do it=1,nt {                    # evolution in time
```

```
                        cshift = cexp(cmplx(0.,-w*it*dt))
                        do ix = 1,nx
                                p(iz,ix,it) = p(iz,ix,it)+q(ix)*cshift
                        }
                }
        }
do it=1,nt
        do ix=1,nx
                call rite(outfd, p(1,ix,it), nz*4)
call exit(0)
stop;           end
```

You are encouraged to make your own versions of this program by editing the file. Here are a few actions provided for you. To carry out the command, press the middle mouse button in the appropriate box:

Make changes in the source code of the program: make **Edit Program** cake edit$_e xample$

Produce executable version of the program: make **Compile Program** cake compile$_e xample$

Perform the depth extrapolation: make **Run Program** cake run$_e xample$

View the (x, t)-section at the first depth step: make **First Section** cake first$_e xample$

View the (x, t)-section at all calculated depths, by running a movie:

After having understood the program's algorithm, you can find out what effect certain parameters have on the downward extrapolation. An obvious parameter is the number of frequencies used in this extrapolation. The data in Figure 1 are very narrow-banded, by default only two frequencies are used. Instead of using the editor and changing the variable nw in the source code, which implies recompiling, you can use the button . This action will bring up a menu with a small panel of parameters which you can adjust. The program will automatically run with you settings and you will be able to compare your choice with the original figure.

In case you want to restore the original versions of figures and programs you can do . This will copy the original versions from the restore directory into the current directory.

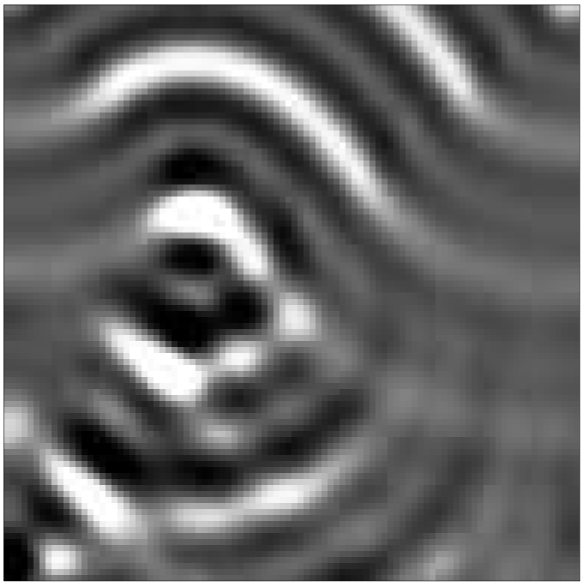

Figure 1: This is the (x, t)-section at the first depth level. To regenerate the figure using the newest version of the program hit the box following the caption: dave2-figure [NR]

The TEX file

```
\def\figdir{./Fig}


\subsection{Exercise: \qquad($\omega$,x,z)-Wavefield Extrapolation}

This document is an actual exercise taken out of Jon Claerbout's book
{\em  Imaging the Earth's Interior}. This exercise performs a wavefield
extrapolation in ($\omega$,x,z)-space. At the top boundary data are
given as if recorded from a spherically diverging wave coming from
within the medium. The goal of this exercise is to show that the
($\omega$,x,z)-space can be used for downward extrapolation of
recorded data. The following is a listing of the existing program written
in ratfor. It is directly included from the directory in which this document
is stored.

\listing{example.r}

You are encouraged to make your own versions of this program by editing
the file. Here are a few actions provided for you. To carry out the
command, press the middle mouse button in the appropriate box:

Make changes in the source code of the program: \syscall{\bf Edit Program }
{cake edit_example}

Produce executable version of the program: \syscall{\bf Compile Program}
{cake compile_example}
```

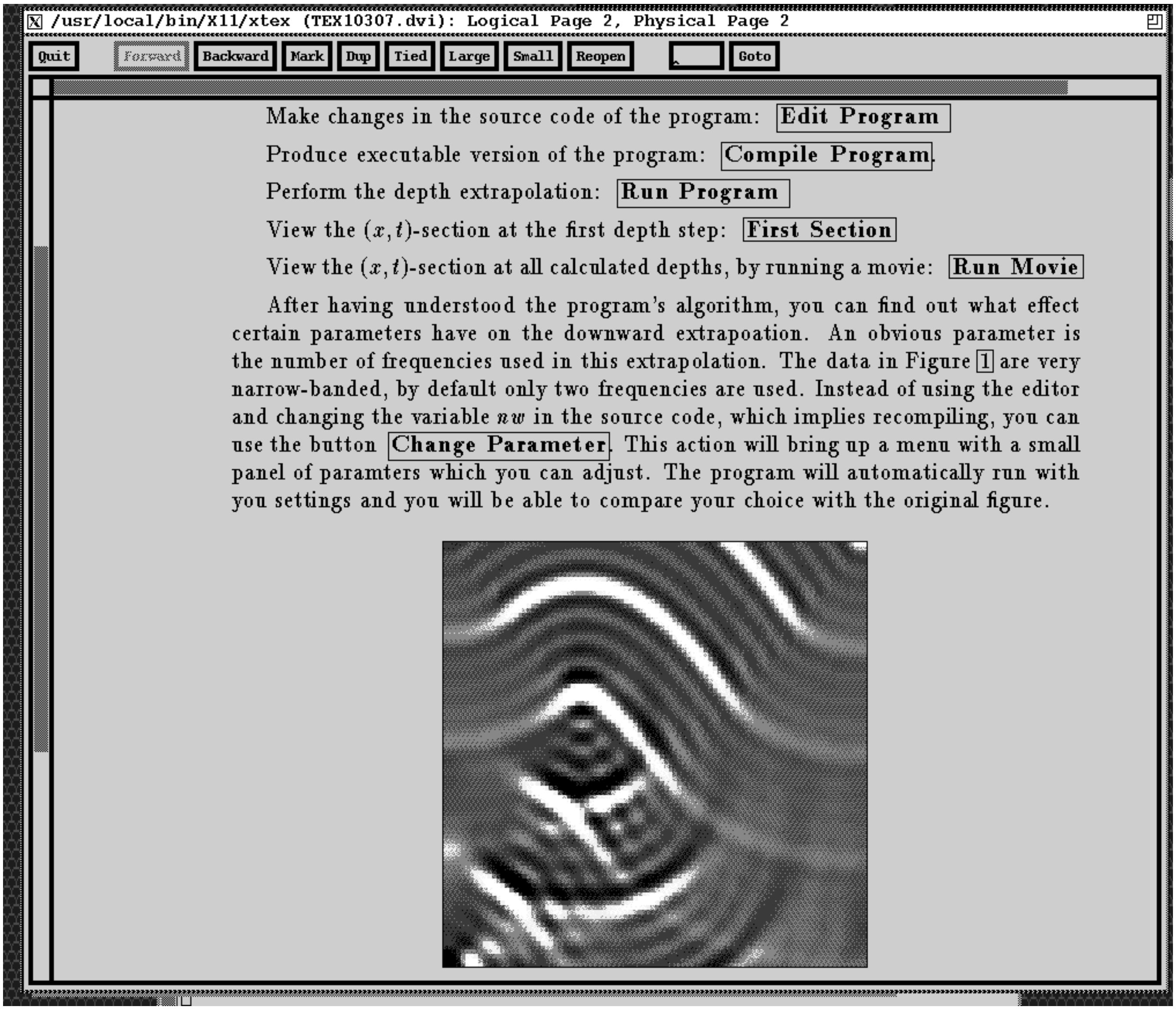

Figure 2: This figure shows the xtex previewer in use. Only the basic text and figures are visible on this screen. Note the presence of several buttons on the screen. dave2-dump1 [NR]

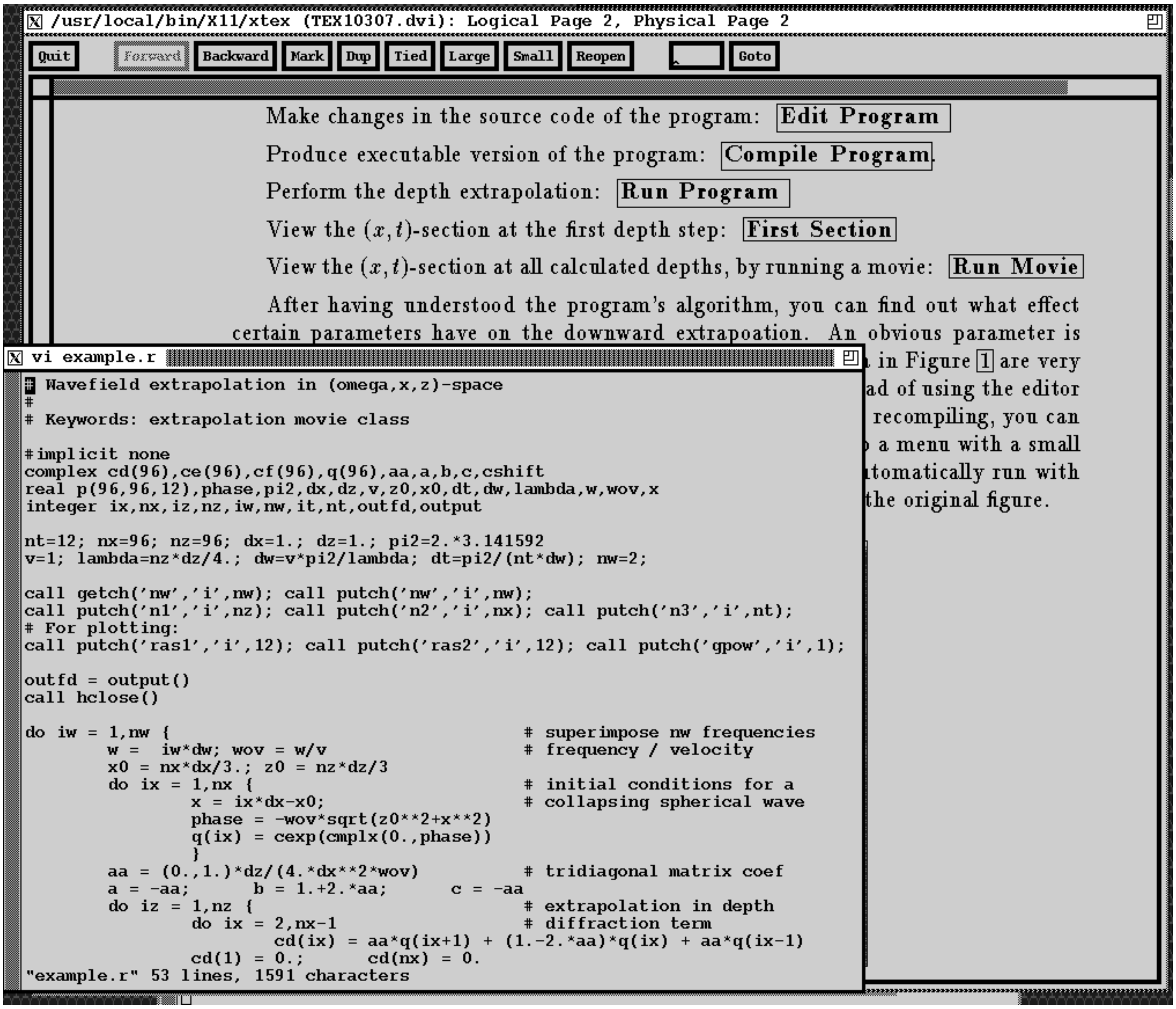

Figure 3: When the *Edit Program* button is pressed a window appears on the screen to allow the user to edit the ratfor program. After the editing session is complete the user can compile the program by pressing the *Compile Program* button. Note that actually the *Compile Program* and *Run Program* buttons are redundant. The dependencies specified in the Cakefile will ensure that the program is recompiled and rerun if necessary. The buttons are included so that the user can run each action in isolation for testing purposes. dave2-dump2 [NR]

Figure 4: When the program has been successfully compiled the user can run the program to create a synthetic dataset by pressing the *Run Program* button. The user can the see the results by pressing either the *First Section* or *Run Movie* button. In this example the *Run Movie* button has been pressed and the user has a movie of the data visible on the screen. dave2-dump3 [NR]

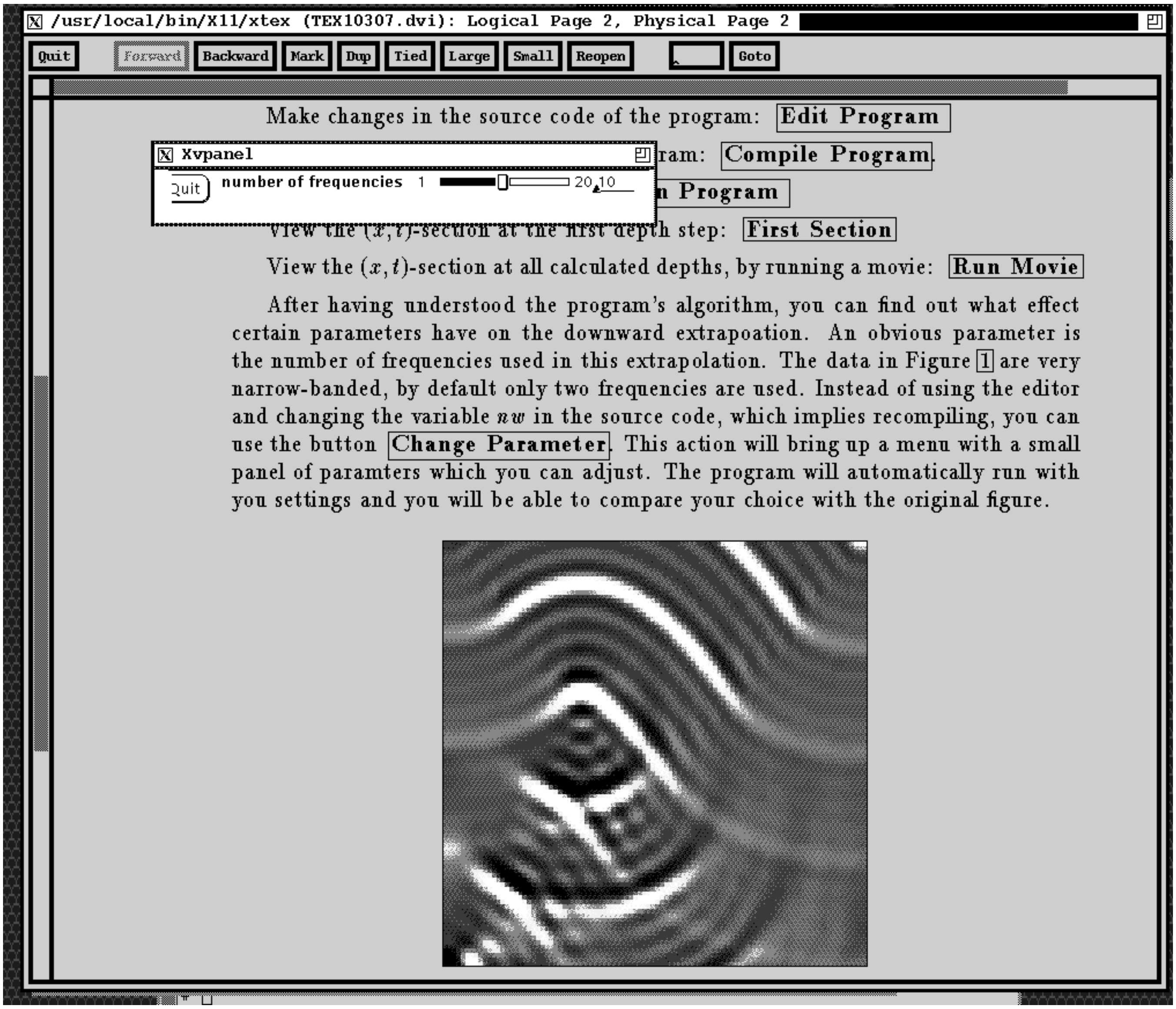

Figure 5: Another possible method of interaction is to run a command that prompts the user for a processing parameter. If the *Change Parameter* button is pressed the *Xvpanel* program described by Steve Cole in this report is invoked. It has been set up to prompt the user for the number of frequencies to use in the modeling. The user has chosen ten frequencies in this example. dave2-dump4 [NR]

Figure 6: The user can press the *figure* button in the plot caption to view the new version of the figure. Note that because the dependencies in the Cakefile include the parameter file modified in the previous example the program is rerun using the new parameter, ten frequencies instead of the default of two. The changes are clearly visible. dave2-dump5 [NR]

```
Perform the depth extrapolation: \syscall{\bf Run Program }
{cake run_example}

View the ($x,t$)-section at the first depth step: \syscall{\bf First Section}
{cake first_example}

View the ($x,t$)-section at all calculated depths, by running a movie:


After having understood the program's algorithm, you can find out what
effect certain parameters have on the downward extrapolation. An obvious
parameter is the number of frequencies used in this extrapolation.
The data in Figure \ref{fig:figure} are very narrow-banded, by default only
two frequencies are used. Instead of using the editor and changing the
variable $nw$ in the source code, which implies recompiling, you can use
the button . This action
will bring up a menu with a small panel of parameters which you can adjust.
The program will automatically run with you settings and you will be able
to compare your choice with the original figure.

\activeplot{figure}{height=3.0in}{.}{This is the ($x,t$)-section at the first
depth level. To regenerate the figure using the newest version of the
program hit the box following the caption:}


In case you want to restore the original versions of figures and programs
you can do . This will copy the original
versions from the restore directory into the current directory.
```

The Cakefile

```
#include <SEP.defs>

read&: example.tex example.r figure.ps
        texpr -s -v -2 example

figure&: figure.ps

edit_example&:
         xterm -n vi -T $EDITOR" "example.r -e $EDITOR example.r

change_parameters&:
        /r3/steve/bin/DEC3100/Xvpanel

compile_example&: example.o ./lib/ctris.o
        FC LDOPTS example.o ./lib/ctris.o SEPLIB  -o example.x

run_example&: example.x
        example.x >Hexample

example.x:  example.o ./lib/ctris.o
        FC LDOPTS example.o  ./lib/ctris.o SEPLIB -o example.x

Hexample:       example.x
        example.x >Hexample

first_example&: Hexample
        Window n3=1 <Hexample |Taplot >tmp
        slice <tmp

Texample: Hexample
        Taplot <Hexample >Texample

movie_example&: Texample
        slice <Texample

figure.v: Texample
        Window n3=1 <Texample |Ta2vplot out=figure.v head=/dev/null

figure.ps: figure.v
        pstexpen figure.v figure.ps


restore&: orig
        cp ./orig/example.r .
        cp ./orig/example.tex .
        cp ./orig/figure.ps .

clean&:
        -RM_CMD *.f *.x *.o  a.out core He* Te* tmp* *dvi *aux *log TEX*


#include <SEP.obj.rules>
```

```
#include <SEP.prog.rules>
```