Hyperbola overlay program development:

a personal engagement

Jon F. Claerbout

ABSTRACT

I wrote a computer program for interactive normal moveout and overlay of hyperbolic and other templates on seismic data. Hyperbolic overlays move and reshape themselves almost continuously as the operator moves a mouse. Benchmarks show only a few seconds are required for data rescaling, zooming and normal moveout. Difficulties I encountered during development were: learning to use the Sunview system, bugs in the Sunview system, and learning a programming style suitable for interactive programs.

INTRODUCTION

The first time I saw the Macintosh home computer running its "Paint" program I realized there is hope of escaping our longstanding frustration in computer graphics. It is hard to define this frustration, but we always feel it while preparing slides for a lecture, figures for a report, or waiting for a job to turn around. The frustration lies in the gap between knowing what can be done, and knowing what we by ourselves can accomplish in a day.

Although it is easy to plot a few wiggle traces on a Macintosh, further investigation showed me the Macintosh is insufficient for serious work in reflection seismology. But the Mac convinced me of the potential of small computers. This year Sun Microsystems joined the SEP and we acquired a SUN 3/160 workstation. It has roughly the capacity of our previous VAX 780, additionally it is built around a large high-resolution color graphics screen, and has a large volume of software supporting graphics interaction.

Around 1970 Dan Tudor taught me how to interpret seismic data with the aid of hyperbolic curves printed on transparent overlays. I made a stack of overlays for various velocities, and I still use them. There are frustrations here that I won't recount, but the resulting interpretations more than justify the effort. This paper is an account of my experience computerizing the hyperbolic overlays.

Rick's movie program

With Rick Ottolini's movie program on the VAX we easily confirmed much old wisdom about seismic plotting: The vertical exaggeration is important. The "clip" value is also important. We learned some new things too: Screens are more linear than hard copy and weak events are more likely to be visible on a screen than on hard copy. "Before versus after" comparisons are much more effective with a movie program. Color can be deceptive and should used cautiously.

Rick's movie program is a huge success. We have made about forty thousand movies. We use it in our research, in our seminars, for debugging, and we hauled Movie in the SUN up to our annual sponsor meeting in May. Although it is a huge success there are still many limitations. Rick's movie program is used this way: A user program puts out a two or three dimensional matrix. In a minute or so, another program (Taplot) converts the floating point values to brightness pixels (bytes). The movie (a cube of bytes) is saved on disk until you are ready to view. Then you invoke Rick's movie program. You can view the volume in many ingenious ways. But Rick's movie program is "interactive" only in the viewing of the results, it is not interactive with the user program that created those results. Rick did not provide subroutines for users to link with their programs. Also Rick's movie program is so massive that no one has has the courage to install his/her own routines in it. So, if you need something extra, timing lines, another color scheme, hyperbolic overlays, an overlay to help identify uniform reverberation, or interactive perturbation of the moveout function, what do you do?

Enter the Sun

After the SEP spring meeting was over and after the master's degree students left town, I noticed the Sun workstation sitting off in a corner. Rick ported his movie program to it (as well as 4-5 other devices in our lab). People did occasionally use the Sun for Rick's movie program, but nobody did anything else with the Sun, although its UNIX (trademark AT&T) software is the same as that of our other two computers. The ethernet does a good job of linking the three computers, but users find it slightly easier to leave all files in the Convex and use its attached RasterTech for movies. I was

pleased that Sun had joined our group and that we had access to this new "high tech" product, never-the-less I didn't want to require anyone to do anything with it. So I decided to spend a week with it myself. With a little coaching from Rick, within about a week, I had a mini-movie program of my own. It didn't have desirable features like zoom and pan, but the frames did move forward under user control, and it did some things with *subroutines in Fortran!* It was no toy either, easily handling five hundred traces (variable brightness) per frame. My initial goal was to document this program clearly so that others in our group could easily build upon it in their own directions. Having met this goal so easily, I thought I would demonstrate a useful application. However, after this initial week of joy followed a month, then another, of struggle, but I think the results are worth it.

TANGIBLE RESULTS

My first one week program "jmovie" is two pages long with a half page fortran subroutine. It is a minimal movie program, plus it identifies the cursor location, and the user can put a yellow mark on the data at the cursor position. The program isn't elegant, but it is a usable starting point for anyone's interactive work.

My big program, "Overlay," is a forbidding 1600 lines. It will be as impenetrable to others as Rick's movie program. Maybe more so. It overlays hyperbolic curves on seismic data and moves the curves around. Some parameters may be selected by dragging a mouse and others are selected by editing numbers on a "control panel." The curves can be unconstrained as on a seismic section, or they can be constrained laterally, as CMP gather curves going through zero offset, or as for a field profile over dipping layers, or over a diffractor. The program also allows for interactive adjustment of the clip (data is kept in floating format), the horizontal zoom, the time datum, the x-origin, and the frame counter. Further, normal moveout takes a few seconds, and you can overlay other curves on the moved out data that assist you in bootstrapping to a depth-dependent interval velocity. There are still more overlays for for identification of multiple reflections on sections, gathers, and profiles.

It is not an exaggeration to say that most viewers find the results "stunning," an observation you should keep in mind if you are reading about my difficulties without seeing the results in person.

PROGRAMMING ENVIRONMENT

You can compute with the Sun's UNIX fortran, but the way to the nifty graphics is through the C language. Fortran is a stronger language in numeric work. Fortran supports complex numbers, and variable dimension statements like "real data(nt,nx)" which C does not. C is a smaller language (hence easier to learn) but it includes pointers, better definition of "scope," and also data structures not in Fortran. But C's greatest strength is its closeness to the operating system and to Sunview, both also written in C. Fortran users are familiar with declarations of reals, integers, etc. In C on the Sun you find the declarations: Frame, Canvas, Panel, Panel_item, Text_subwindow, Pixwin, and other concepts that have been built upon C structures. By means of C-callable subroutines, you create these objects, give them attributes, and change them. C programs can also call fortran programs and vice versa.

Effort calibration

I worked hard on this project with few distractions for three months. The result is pleasing, though far from perfect. Would like to follow my footsteps? Before I recount my trials, I'll tell you a bit about my own programming skills to enable you to calibrate yourself.

I started learning C eight years ago but I always use Fortran whenever I can. I use C mainly to fill in the gaps in Fortran. I am probably a slightly better than average SEP programmer, but this is merely the advantage of more experience (reduced by a weak short-term memory). I'll mention Levin, Clayton, Thorson, Ottolini, Kjartansson, Hale, Ullmann, all of whom are much better programmers than I am.

Slava Trudu !

At first I was rusty in C, but was refreshed in less than a week. Then the real burden was the Sunview manual. There are not enough examples, and what is there is more like a collection of clues, rather than a learner's guide. (A new manual release reached me just after I completed this paper). To be fair to Sun Microsystems, I must confess that they invited me to attend some classes, but I declined. I had hoped for an occasional half day's time from any one of their analysts, but they declined, so I was in a position little different from the average customer.

By the end of the first month I was worse off than at the end of the first week. My program crashed frequently and unpredictably. I resolved to give up, but before I did, I boiled the program down to a tiny example that showed that writing vectors will crash sometime after the following two conditions are present: (1) a color table has been

defined, (2) the vectors require clipping at the boundary. I sent the bug to Sun who soon agreed the bug was real. (That was a relief for me because I found the Sunview description of color particularly abstruse, with no examples). Two months later they say they have a fix that I should receive in the next software release in another month. Knowing how to work around the bug by doing my own vector clipping, I proceeded anew, carefully avoiding this difficulty and a few others. The most frequent diagnostic messages I saw, after compiler syntax, were "core dumped." Also, it is common to crash Sunview getting no diagnostics what-so-ever.

Please do not infer from this that I am unhappy with the Sun product. On the contrary, I am delighted by it. But I want you to understand the difficulties that accompany any product in such an early stage of development.

Creating an interactive environment

Finally comfortable knowing what I could and could not do in Sunview, the next unfamiliar burden I assumed was that of organizing my thoughts to make a program that could be operated like a machine, to be driven like a car, without leaving the operator continuously confused. Historically the programs geophysicists write have an input and an output, and they work from one to the other as fast as they can. My interactive program shows a control panel, and the goal is for the control panel to be simple and self explanatory so people can use it with almost no training, and so they can return to it without wanting to refer to any written material.

When the program is working on NMOed data, panel items must be removed that apply only to raw data. The user may switch back and forth several times between frames and between raw data and moved-out data. Nothing should take longer than necessary. Users will walk away if things start to take more than 20 seconds. Almost anything that touchs *every point on the screen* or *all the data points* will take a second or more, this includes erasing the window, scaling data, zooming data, and doing moveout.

Application of AI: the "make" trick

Any time anyone changes anything, it is possible that many things need to be updated. Typically, it takes about 15 seconds to update everything, so you don't want to update any more than what is required, and that often takes much less. As the program grows in scope, the complexity of possibilitys grows faster, and soon more programmer effort is devoted to thinking about the effect of the *sequence* of user operations than thinking about the operations themselves.

I finally got a solved the sequencing problem, when I applied the bit of AI that is used in the UNIX "make" utility. For each subroutine of the program that consumes much time or memory, I define the inputs and the outputs, and the parameters that affect the outputs. Each time an output is created, it is time stamped. Each input and each adjustable parameter also has a time stamp. At the beginning of any such subroutine, the time stamps are inspected first. If any of the inputs or parameters have time stamps later than that of the output, then the subroutine must be run again, otherwise the subroutine can return without doing anything. With the compute intensive modules in this form, the programmer can return to familiar "straight line" patterns of thinking for the overall structure. Any time the user changes anything, you just change its time stamp and then run the "straight line" program.

BENCHMARKS

Timing and descriptions of some of the principal program components follow: Times are based on a field profile of 90 traces and 900 time points. I would like to have used more time points, but I needed to keep the data all on the screen because of the vector clipping bug, and my inability to get scroll bars to work, and the screen not being rotatable. For the benchmarks, this data was horizontally zoomed by a factor of six, thereby filling about half of the viewable area, the remainder being used by the "control panel" and a space for running and editing the program. Rough timings are:

hyperbola draw	.2s
frame change	.5s
refresh	.5s
moveout	3.5s
new clip	8.0s
zoom	2.5-4.5s
ethernet floats	14s
ethernet bytes	3s

The hyperbola redraw occurs with sufficient speed that you feel it is moving almost continuously across the screen.

If we had hardware floating point, the only time that would speed noticeably would be the changing of the clip. This would be more important on seismic sections (which typically have more traces at lesser horizontal exaggeration).

The moveout correction is surprisingly fast because I do it all in integer arithmetic, and I do the square root by one iteration of Newton's method from a starting guess at a

neighboring point. Interpolation is to the nearest neighbor. While crude, this interpolation is not visually displeasing. Also, I only move the displayed bytes, not the floats themselves.

Most aggravating is the zoom change. Since you are unlikely to know in advance what zoom will be pleasing, you are likely to want to change the zoom several times. Even more annoying is that many of the other stages pass through zoom last, so for example for NMO, the zoom time adds to the NMO time. Rick coded a couple dozen special purpose zoom programs to optimize each possible numerical choice of horizontal and vertical zoom. My program zooms horizontally only, and I handle the nonzoom case specially (2.5s) versus the slower general case (4.5s). Users would like vertical exaggeration too but I didn't code that yet.

The table includes times for reception of the data from the Convex (minisupercomputer) via ethernet at a time when the load factor on the Convex was about 3.5. In the present study, no calculations were done in the Convex, so these times are presented for reference only. If heavier calculations were contemplated, these transmission times would be a relevant constraint.

DISCUSSION

Pessimism

1. Now it runs only on a SUN. Interactive window programming languages will be subjected to rapid developments and the program will need to be revised many times, just to keep it running.
2. Hardware zoom and pan support is needed. (See benchmarks). We have hardware zoom in the RasterTech and the AED's.
3. The screen should be rotatable from its usual "landscape mode" to "portrait mode." This is not only because our traces run 1500 points, but also because we often process by traces and the Sun refreshes the screen by horizontal scan lines.

Optimism

I hope my program becomes popular within the group. I'll test it in some classes. The program should sparkle at this year's SEG exhibition, but I don't expect to wait long before some professional programmer does a better job with this idea. So maybe the effort was misplaced. On the other hand, when my research or teaching takes a new direction, my programs will be quickly made interactive. This kind of programming is

likely to grow in use, and by struggling to be among the first, I have not done a great deal more work than others will need do later. (Meanwhile, I'll be prepared to teach them!)

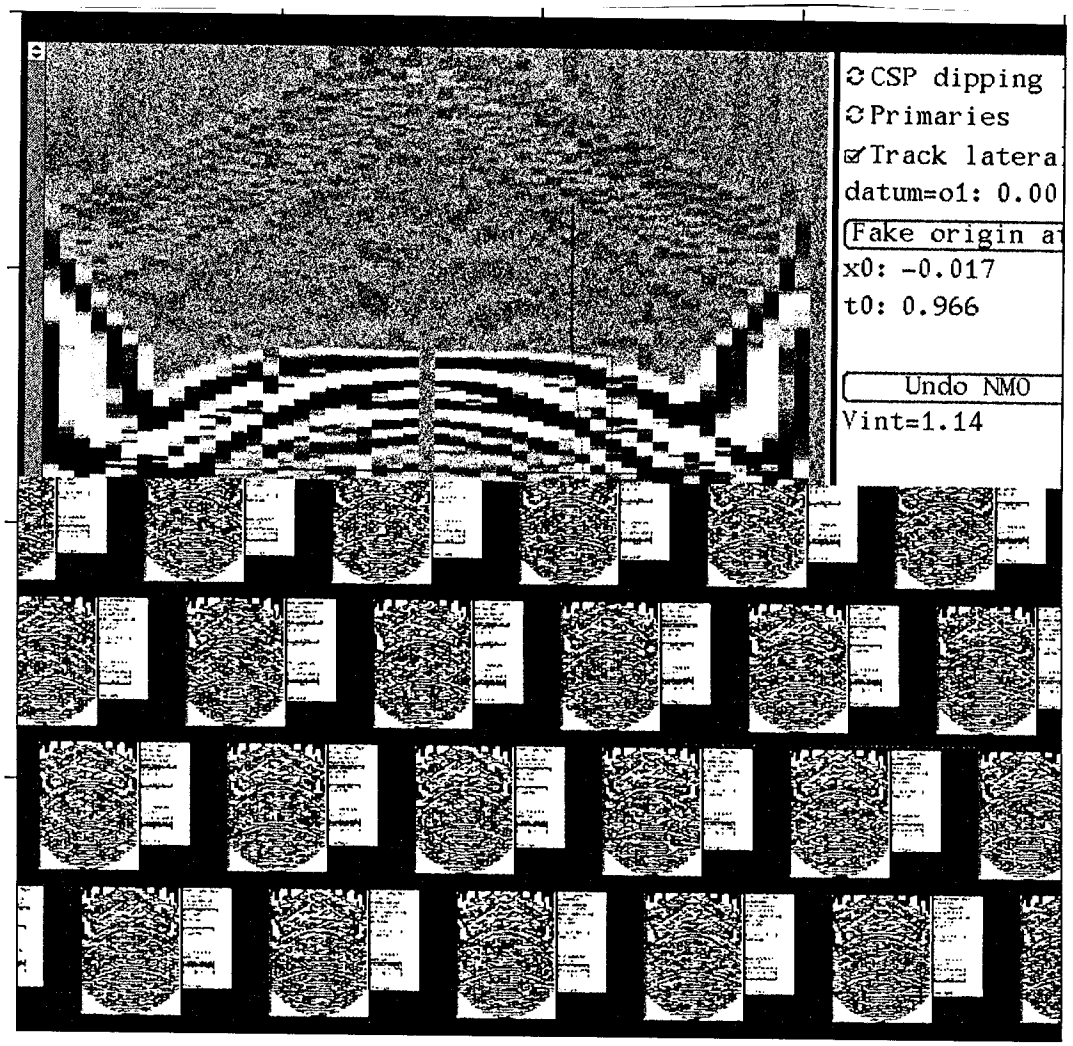


FIG. 1. Attempted screen dump of running "Overlay" program. Upper left: Deep marine split-spread profile after normal moveout. Superposed slightly right of center are interval and RMS velocity profiles. On the actual screen these are in color. Upper right: Top portion of "control" panel. The lower half of the figure is presumed to show a bug in the screen dump software. It seems to be about 25 small replicas of the entire screen.

Your access to my program

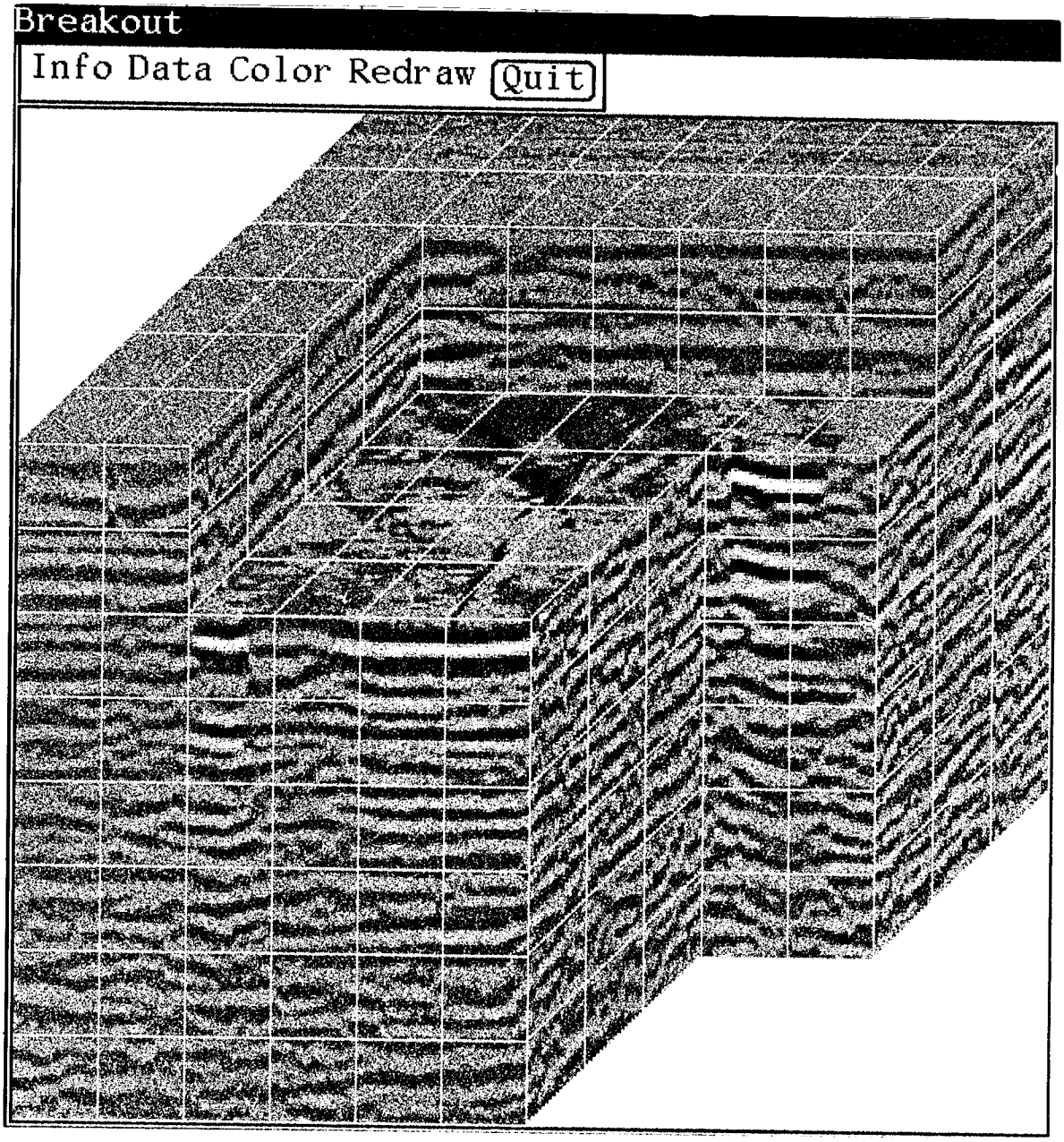
My program will be on display at the November SEG meeting in the booth of Sun Microsystems. It is available now without license or charge to all sponsors of the Stanford Exploration Project.

Conclusion

The Sun workstation provides a data interpretation environment beyond any other that I have ever seen. (I am not competent to compare the Sun to other products in its class). To take advantage of this new environment, programmers have many hurdles to leap.

REFERENCES

Ottolini, R., 1984, Seismic Movie Machine, *Geophysics,* February



An image of one of the cube sculpturing tools written for the Sun workstation. -R.O.