What's NeXT? Interactive seismic graphics applications on a NeXT workstation

Rick Ottolini

ABSTRACT

I coded a multiple reflection tutorial and seismic data display program for the new NeXT interactive graphics workstation. The motive was to investigate NeXT's claim that it is easier to write software for their computer. NeXT software has solved some applications writing problems at SEP, but currently lacks essential features such as FORTRAN and good image software.

INTRODUCTION

A pressing software need at SEP is an easy way to write interactive graphics software for education, data display, and interactive processing. One of the new graphics workstations, the NeXT[1] computer, was specifically designed to be easy to write interactive graphics for. The NeXT workstation is interesting because it incorporates as standard a number of new features not standard on other workstations such as a huge optical disk, array processor chip, high resolution grayscale monitor, UNIX, EtherNet, same screen and printing graphics language, powerful program writing utilities, and PC-type applications under UNIX.

The following properties are important for developing interactive seismic graphics software. NeXT workstation is graded for each property on a scale from *A* to *E* where *A* is much better than average, *C* is average, and so on.

- *Expression of scientific algorithms—C.* FORTRAN has better numerical abilities than C, but is missing on the NeXT.

[1]NeXT Inc., Palo Alto, CA

- *Ability to display a seismic image—B.* A seismic image is rendered on a graphics terminal as an array of colors, where color represents amplitude. The NeXT screen is four-level grayscale. Dithering gives the appearance of more gray levels, but is unsatisfactory for seismic images (Figure 2). There is almost no control over how images are dithered and scaled.

- *Storage of seismic databases—A.* A huge, rewritable optical disk is standard. Remote file access through network file system is well supported.

- *Ease of writing applications—B.* NeXT rates a *D* for having to learn three new computer languages before starting: Objective-C—a superset of C, Display PostScript—an interactive version of the PostScript graphics language, and the Interface Builder—designs user interfaces by drawing them. However, once learned, the NeXT is one of the easiest computers to write interactive graphics for. (I wrote the examples in this article without having to take the course NeXT teaches or ask NeXT for technical advice.)

- *User interface toolkit—B.* NeXT has a nearly complete and powerful set of functions. The display is rather handsome (Figures 1 and 2). Sometimes it is hard to find a particular function due to the method NeXT organizes code (object-oriented inheritance trees).

- *Printing—A.* Hardcopy figures are important for courseware and scientific publications. The same graphics language is used for the screen and printing—machine independent PostScript.

- *Speed—C.* NeXT has about the same number crunching and compilation speed as a Sun-3 or Mac-II despite a faster CPU chip (68030). Much of the speed increase is consumed by Display PostScript. The optical disk slows the start of programs and data transfers. The digital signal processor was not available for testing at this time.

- *Compatibility—C.* Interactive graphics applications written for a NeXT computer don't work on other computers because no one else implements Display PostScript—yet. The UNIX, C, Objective-C, and PostScript languages are pretty much the same as on other computers.

SEISMIC EXAMPLES

Two simple seismic applications investigate the capabilities of NeXT. The first is a multiple reflection tutorial originally written by Claerbout on a Sun-3 (Claerbout, 1989). It has three control buttons and two drawing windows—not too simple and not too complicated (Figure 1). The second application displays a seismic data image (Figure 2). It studies file and image display capabilities.

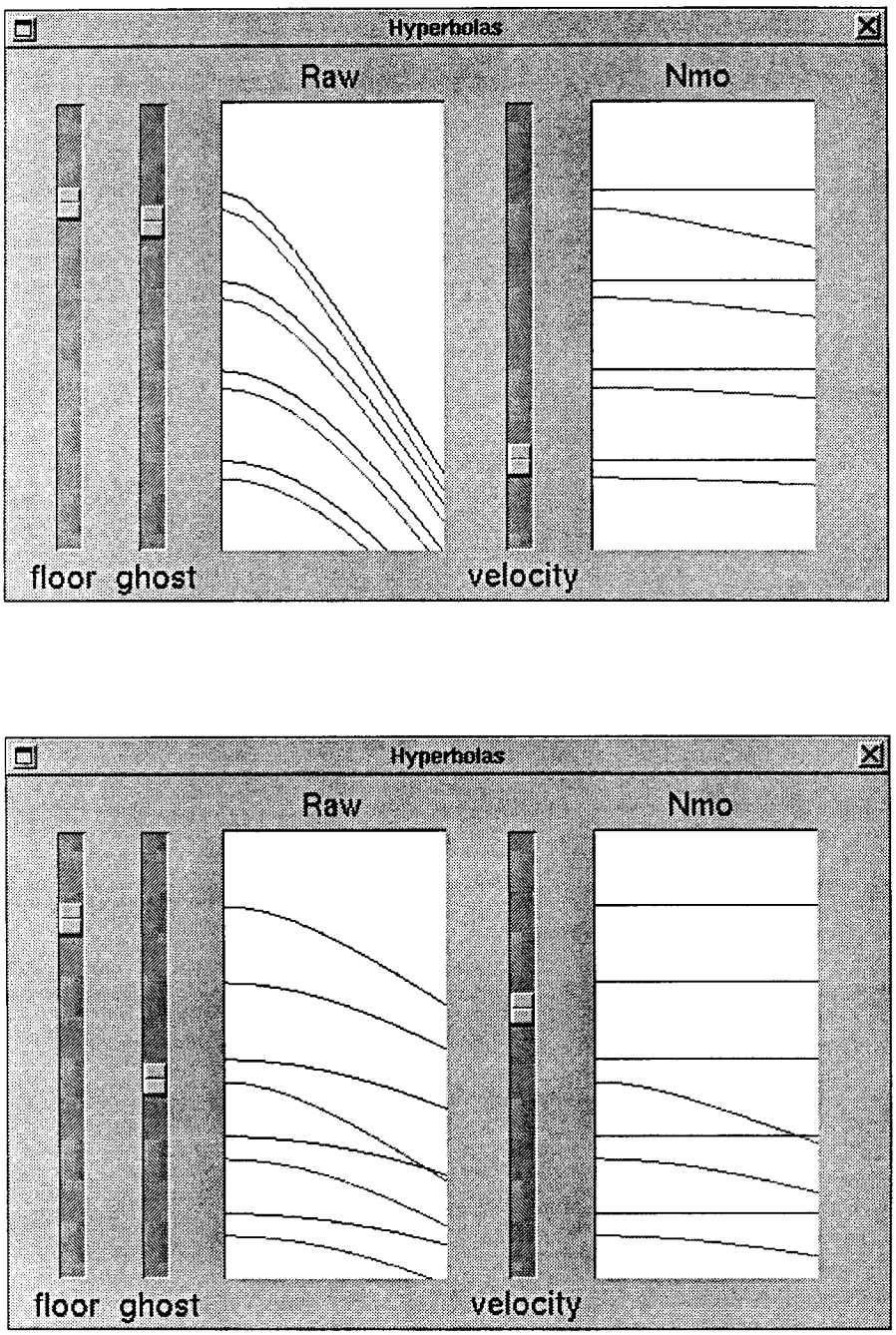

FIG. 1. Sample application # 1: This multiple tutorial illustrates effect of the seafloor depth, ghost interval, and velocity on seafloor multiples (black) and ghost multiples (gray). Adjusting the sliders changes the two images. Details described in text. Adapted from Claerbout (1989).

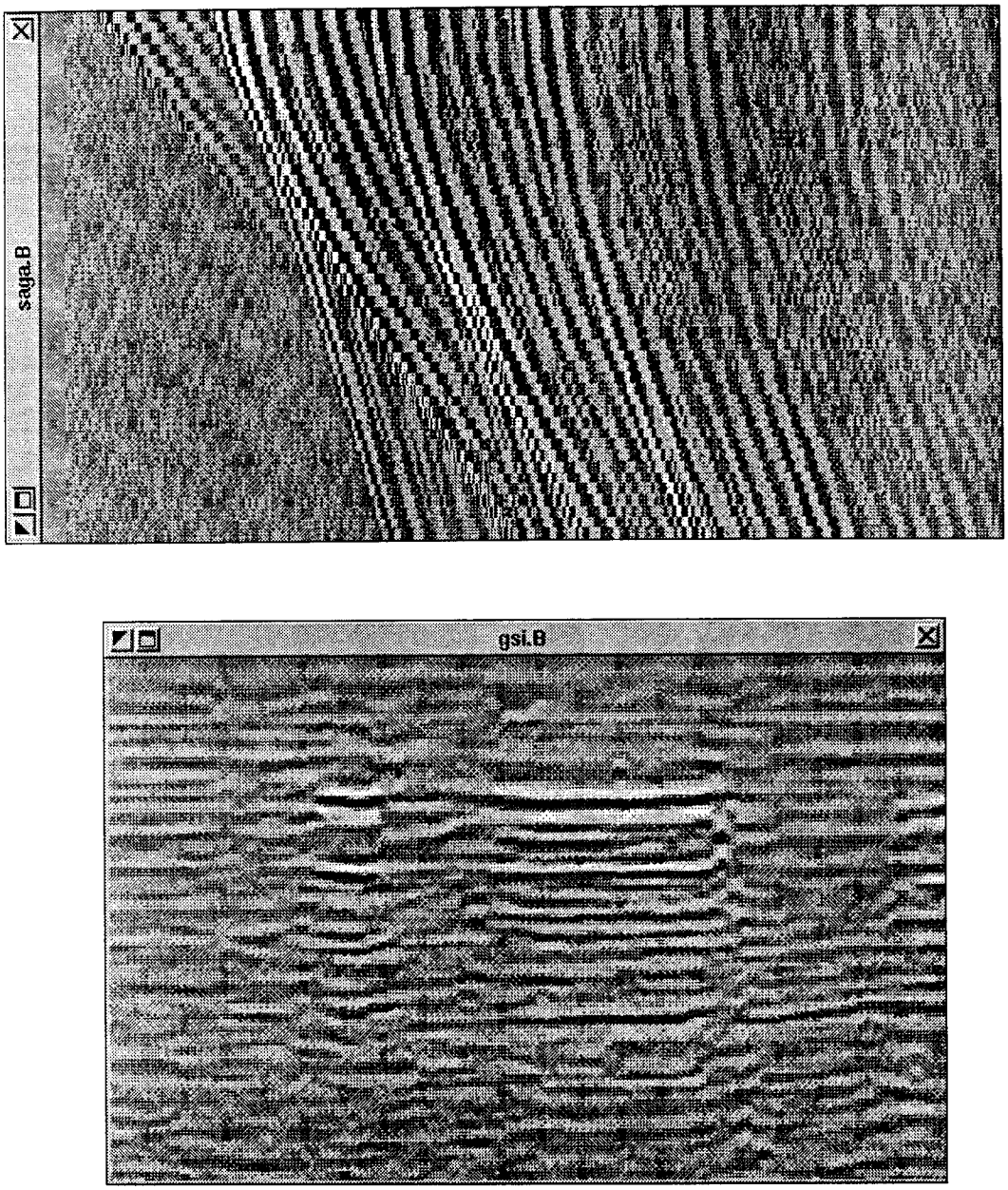

FIG. 2. Sample application #2: seismic data viewer. Written to study the manipulation of seismic databases and the quality of seismic displays.

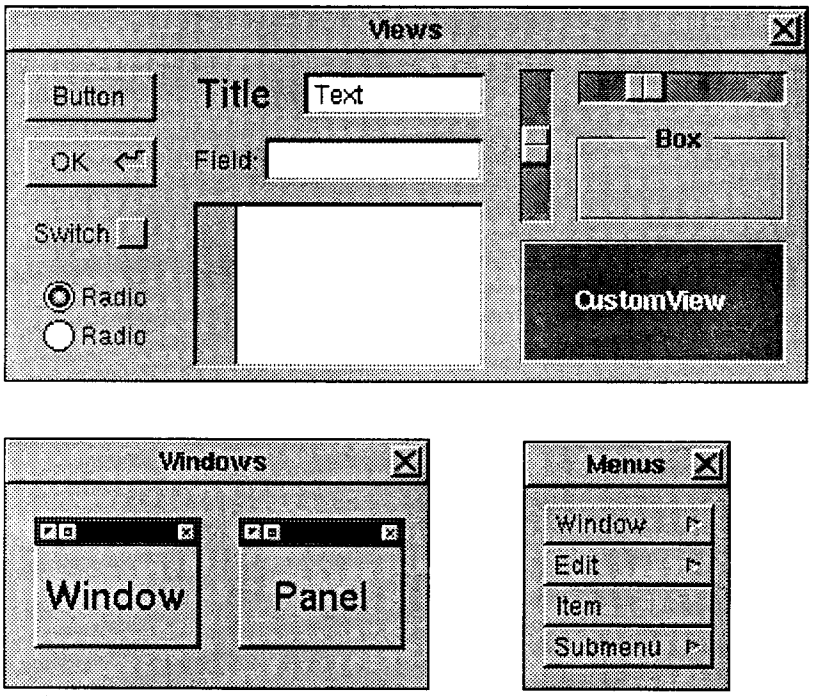

FIG. 3. Part of the NeXT Interface Builder interface construction kit. One selects, copies, adjusts, and re-labels items to construct the custom interface, such as that of Figure 4. (This image is copyright by NeXT.)

Program construction

Programs are written in two parts—an interface part and a custom part. The programmer first draws the interactive interface—windows, menus, buttons—using a toolkit called the Interface Builder (Figure 3). Then the Interface Builder generates C language code for running the interface plus a code skeleton for the programmer to insert custom parts. The custom part implements the 'science' of the program—a numerical model and its graphical depiction.

Multiple reflection tutorial example

Lets us walk through a simple multiple reflection tutorial (Figure 1). Three sliders control the position of various reflection events. The *floor slider* controls the time of the inner offset of the seafloor. The *ghost slider* controls the time of the inner offset of the source reverberation. The seafloor-to-ghost interval is positive and stays the same size unless the ghost slider is moved. The *velocity slider* controls the time of the outer offset of the seafloor. The sliders input and display travel times which are converted to floor, ghost, and velocity variables within the program.

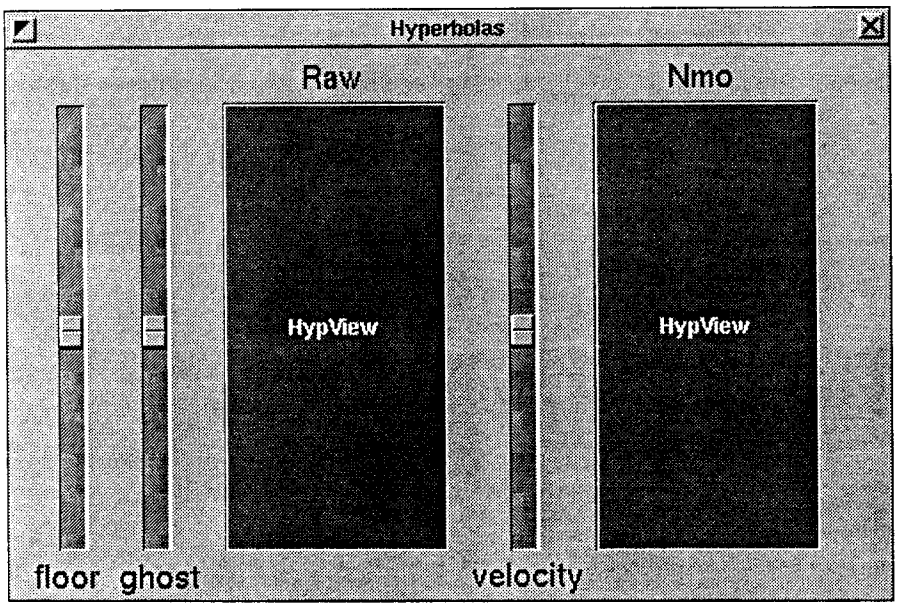

FIG. 4. The multiple tutorial interface constructed using the Interface Builder toolkit of Figure 3. The programmer supplies additional code to react to interface controls and draw the contents of the two HypView boxes.

Reflection events are displayed in two output windows—the raw events and moveout corrected at the seafloor velocity. The seafloor and its multiples are in black; the ghost and its multiples in gray. The ghost and seafloor multiples have different moveout behavior (Figure 1).

The first step in writing the this application was is to draw the interface. We begin with an empty window and the toolkit of Figure 3. One grabs interface items from the toolkit, positions and sizes them, and adds labels. This application used three sliders, two custom views, and five labels (Figure 4). Menus and auxiliary windows are created in the same manner.

Next the programmer specifies how the various parts of the interface interact with each other and custom parts of the program. This is done using the graphical metaphor of a wiring diagram (Figure 5). Each object has input and output slots. For example, the input slot of a slider is a subroutine that sets the slider position. An output slot is the hyperbola view.

After drawing the interface and connections, the programmer clicks a button to generate source code. The code is output in Objective-C a variant of the C programming language for writing modular code. For clarity, only the C language equivalent will be shown in the examples.

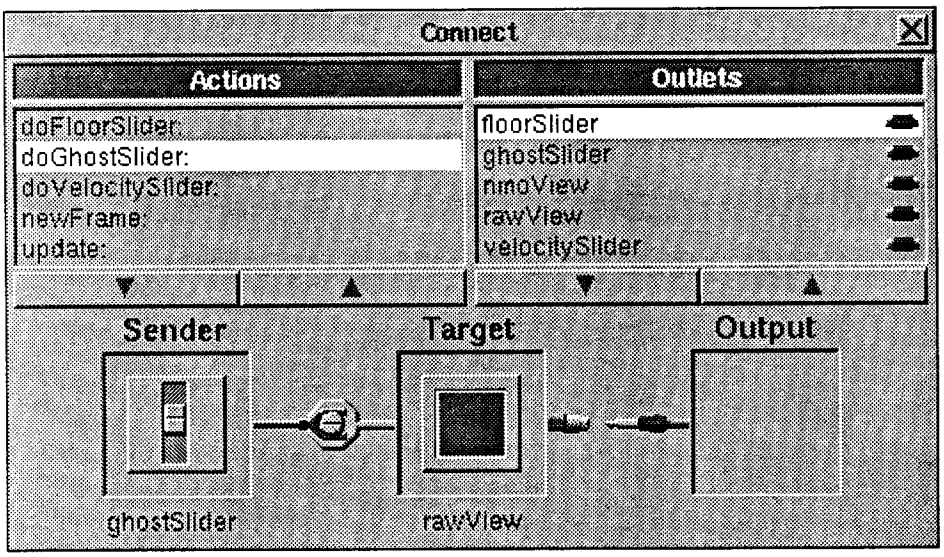

FIG. 5. Interactions between various parts of the interface are specified using this graphical wiring metaphor that is part of the Interface Builder. Actions are names of input subroutines while outlets are names of output variables. Some of these are supplied by the predefined interface objects of NeXT application toolkit (Figure 3) and others are written by the programmer. This figure shows programmer-written actions for the hyperbola views of Figure 4. Specifically, this illustrates that the ghost slider has been connected to the HypView code module via the doGhostSlider subroutine. (The format of this figure is copyright by NeXT.)

Several pieces of code are produced—the interface, the main application, a skeleton of the custom part, and a compilation file. In most situations, the programmer need only fill in the custom part skeleton.

NeXT code structure follows the modular strategy of Ottolini (1986). Code is organized about data structure, called objects. Modules are divided into an interface part and implementation part. Figure 6 is the interface code skeleton generated for the custom part of the multiples example. The skeleton only knows about connections to the interface. The programmer adds variables and subroutines describing the scientific and custom graphics parts (Figure 7). Figures 8 and 9 show the skeletal code and programmed additions for the implementation part of the program.

In summary, the programmer programs a NeXT application by first drawing the application interface on the screen. Then the Interface Builder will generate most of the interface code which the programmer writes the non-interface parts.

```
/* interface file generated by the NeXT Interface Builder */

#include <appkit/appkit.h>

typedef struct HypView
[
        VIEW_PARTS      /* variable list inherited from View and its parents */
        id      nmoView;        /* interface object links */
        id      rawView;
        id      floorSlider;
        id      ghostSlider;
        id      velocitySlider;
]

/* HypView talks to these objects */
extern id       setNmoView (id anObject);
extern id       setRawView (id anObject);
extern id       setFloorSlider (id anObject);
extern id       setGhostSlider (id anObject);
extern id       setVelocitySlider (id anObject);

/* these objects talk to HypView */
extern id       doFloorSlider (id sender);
extern id       doGhostSlider (id sender);
extern id       doVelocitySlider (id sender);
```

FIG. 6. Interface code skeleton automatically generated by the Interface Builder. (Has been translated from Objective-C to pure C by the author for explanation purposes.) The variables and subroutines are connections to the interface designed by the procedure shown in Figure 5. The programmer will then customize this code.

Seismic data viewer example

The purpose of the seismic data viewer is see how well NeXT can render seismic data as an image. An image is a rectangular array of numbers, in this case seismic amplitudes, displayed as color. Since the current NeXT screen has only four colors—black, white, light gray, and dark gray—dithering is used to simulate a larger color range. Results are shown in Figures 2.

The image graphics command of the NeXT is the *image* command of Display PostScript. It takes an array of numbers and dithers it into grayscale. The dithering algorithm is much less sophisticated than those discussed by Cole and Dellinger (1987) and in my opinion, leads to less than optimal, although usable, seismic displays. It takes a couple of seconds for the NeXT to go from input data to dithered image. Once rendered, the image can be saved for rapid animation.

This application was built using the Interface Builder as described in the previous section. In this case, another set of control objects were used—popup panels for reading seismic datasets and menu commands to invoke these panels. The design was straight forward.

```
/* interface file modified by programmer */

#include <appkit/appkit.h>

/* initial values defined by programmer */
#define HYP_MODE_NMO            0
#define HYP_MODE_RAW            1
#define INITIAL_VELOCITY        1.
#define INITIAL_FLOOR           .5
#define INITIAL_GHOST           .1
#define INITIAL_XMIN            0.
#define INITIAL_XMAX            2.
#define INITIAL_NX              10
#define INITIAL_TMIN            0.
#define INITIAL_TMAX            2.5
#define INITIAL_MODE            HYP_MODE_RAW

typedef struct HypView
{
        VIEW_PARTS      /* variable list inherited from View and its parents */
        id      nmoView;        /* interface object links */
        id      rawView;
        id      floorSlider;
        id      ghostSlider;
        id      velocitySlider;
        float   velocity;       /* parts added by programmer */
        float   floor;
        float   ghost;
        float   xmin;
        float   xmax;
        int     nx;
        float   tmin;
        float   tmax;
        int     mode;
}

/* HypView talks to these objects */
extern id       setNmoView (id anObject);
extern id       setRawView (id anObject);
extern id       setFloorSlider (id anObject);
extern id       setGhostSlider (id anObject);
extern id       setVelocitySlider (id anObject);

/* these objects talk to HypView */
extern id       doFloorSlider (id sender);
extern id       doGhostSlider (id sender);
extern id       doVelocitySlider (id sender);

/* subroutines added by programmer */
extern id       drawSelf (NXRect* rects, int n);
extern id       newFrame (NXRect* frameRect);
extern id       setMode (int mode);
extern id       update (float floor, float ghost, float velocity);
```

FIG. 7. Programmer customized interface code from Figure 6. The programmer has added the scientific model, this case hyperbolas, and custom graphics.

```
/* implementation file generated by the NeXT Interface Builder */

#include "HypView.h"


id setNmoView (self,anObject)
id self;
id anObject;
{
        nmoView = anObject;
        return (self);
}

...

id setFloorSlider (self,anObject)
id self;
id anObject;
{
        floorSlider = anObject;
        return (self);
}

...


id doFloorSlider (self,sender)
id self;
id sender;
{
        return (self);
}

...
```

FIG. 8. Part of the implementation code skeleton generated by the Interface Builder. The first two subroutines, setting up the connections, are complete. The last subroutine shown needs more code.

CONCLUSIONS

It is fairly easy to write interactive seismic graphics applications on the NeXT after overcoming the initial hurdle of learning new computer languages. Seismic image display could be improved. Improvements will nodoubtless occur as early models of NeXT evolve.

ACKNOWLEDGMENTS

I thank the Stanford Academic Instructional Resources laboratory for access to the NeXT workstation and NeXT developer consultants Ali Ozer and Paul Hegarty for advice.

REFERENCES

Cole, S., and Dellinger, J., 1987, Display of grayscale data on a bilevel output device: SEP-**51**, 377.

Claerbout, J.F., 1989, Interface for system independent plotting: SEP-**60**.

```
/* implementation file modified by programmer */

#include "HypView.h"


id setNmoView (self,anObject)
id self;
id anObject;
[
        nmoView = anObject;
        return (self);
]

...

id setFloorSlider (self,anObject)
id self;
id anObject;
{
        floorSlider = anObject;
        /* added by programmer */
        setMinValue (floorSlider,-tmax);
        setMaxValue (floorSlider,-tmin);
        setFloatValue (floorSlider,-floor);
        return (self);
}

...

id doFloorSlider (self,sender)
id self;
id sender;
{
        /* added by programmer */
        float t;

        floor = - floatValue (sender);
        t = - floatValue (velocitySlider);
        if (t != floor)
                velocity = sqrt (xmax*xmax / (t*t-floor*floor));
        setFloatValue (ghostSlider,-(float+ghost));
        display (self);
        update (nmoView,floor,ghost,velocity);
        return (self);
}

...

/* added by programmer */
id newFrame (rectFrame)
NXrect* rectFrame;
{
        id self;

        self = newFrame (super,frameRect);
        velocity = INITIAL_VELOCITY;
        floor = INITIAL_FLOOR;
        ghost = INITIAL_GHOST;
        xmin = INITIAL_XMIN;
        xmax = INITIAL_XMAX;
```

FIG. 9. Part of the implementation code customized by the programmer. In this case there were three modifications: (1) slider initialization; (2) code to process slider changes; and (3) view initialization.

```
/* interface file generated by the NeXT Interface Builder */

#import <appkit/appkit.h>
#import <appkit/View.h>

@interface HypView: View
{
        id      nmoView;
        id      rawView;
        id      floorSlider;
        id      ghostSlider;
        id      velocitySlider;
}

- setNmoView: anObject;
- setRawView: anObject;
- setFloorSlider: anObject;
- setGhostSlider: anObject;
- setVelocitySlider: anObject;
- doFloorSlider: sender;
- doGhostSlider: sender;
- doVelocitySlider: sender;

@end
```

FIG. 10. Objective-C version of interface code skeleton actually produced by the Interface Builder. This is the Objective-C equivalent of the code in Figure 6. Syntax variations are explained in the Appendix.

Ottolini, R., 1987, Techniques for organizing interactive graphics programs: SEP-**51**, 421.

APPENDIX: NOTES ON NEXT PROGRAMMING LANGUAGES

Objective-C

Objective-C is a superset of the C language, written by StepStone of Connecticut, to make it easier to write modular, object-oriented programming. The motivation and methodology of this coding style is discussed by Ottolini (1986). In brief, code is organized about shared data structures called objects. An interface module describes the data structure and how it interacts with other objects. A implementation module is the code that implements the actions described in the interface.

Objective-C appears syntactically different from C in a half-dozen ways, although they are mostly the same. The main effect is to impose good organization in C programs. Figure 10 is the Objective-C equivalent of the C code of Figure 7 and is what is actually produced by the interface builder. The syntax differences include:

- The construction *@interface NAME : PARENT* declares a data structure of that name containing all of the variables and subroutines of its parent. In this

example, HypView customizes the general graphical view object by adding information about hyperbolas.

- *#import* means to include other code interfaces.
- Subroutine declaration and calling have a different order. The form is *+-name: (argtype1)argvalue1 argname2: (argtype2)argvalue2 ....* A plus sign means allocate a new object while a minus sign means manipulate the internal variables. This syntax allows successive subroutine calls to be appended on code line in the manner of UNIX pipes. In ordinary C, successive subroutine calls on the same code line would be tangled in a nest of parentheses.
- Objective-C allows different subroutines and variables to have the same name as long they apply to different objects or different arguments. For example, almost every object has the subroutine 'new' for creating a new instance of that object. Some have a second 'new' that takes a rectangle for an argument and draws the new object into that rectangle.

The use of parents and synonyms makes programs more clear, concise and easier to write. The new subroutine calling sequence takes a while to get used to and probably isn't worth the change.

The main competitor to Objective-C is C++. They are similar in function, but different in syntax. Objective-C more closely resembles a predecessor object-oriented language called SmallTalk while C++ looks more like C. C++ does more work at compile time while Objective-C does more work at runtime. The Objective-C approach permits separately runable modules, such as the interface and custom parts of the program. The C++ approach catches bugs earlier. I believe commercial acceptance will determine a winner, with C++ currently leading.

Display PostScript

Display PostScript is a superset of page PostScript for interactive graphics terminals. It adds interactive input and multiple separate graphics canvases that can change with time. It also defines a binary or compiled equivalent to ascii PostScript to increase speed.

NeXT allows either native PostScript code or C-subroutine equivalents. Native code is faster, though this mixture of C and PostScript is more confusing.

Figures 1-5 in this paper are PostScript bitmap images directly from the NeXT screen, transferred to the SEP LaserWriter, and copy-machine reduced. For better line-drawing fidelity than screen dumps, it is not too difficult to connect the application program to the NeXT operating system print manager. The print manager

requests each graphics object in a program to draw itself and sends the output to a file or printer rather than the screen. [2]

[2]Objective-C is a trademark of StepStone. Display PostScript and PostScript are trademarks of Adobe. NeXT, NeXT Step, and the Interface Builder are trademarks of NeXT. Network File System and Sun-3 are trademarks of Sun. UNIX is a trademark of AT&T. EtherNet and SmallTalk are trademarks of Xerox and ParcPlace. LaserWriter and Macintosh II are trademarks of Apple.