BOXER: A RECONSTRUCTIBLE COMPUTATIONAL MEDIUM

Programming is most often viewed as a way for experts to get computers to perform complex tasks efficiently and reliably. Boxer presents an alternative image—programming as a way for nonexperts to control a reconstructible medium, much like written language, but with dramatically extended interactive capabilities.

ANDREA A. diSESSA and HAROLD ABELSON

Writing is an everyday activity for most people in our society—whether it be in the form of a list, a letter, or a scribbled note in the margin of a book—even though few possess expert writing skills. Within a generation, programming will also be a part of the everyday lives of many people who do not have expert programming skills. Naturally, popular programming languages will differ from current general-purpose computer languages, which are designed primarily for programming professionals. Indeed, the very idea of what it means to "program" will change as we come to recognize that, as with writing, the significance of programming derives not only from the carefully crafted works of a few professionals, but also from the casual jottings of "ordinary" people.

This article presents a view of what programming could be like as a common everyday activity for most people. The central image is that of controlling a reconstructible medium, much like written language, but with dramatically extended interactive capabilities. We begin with some general observations about programming in this context and continue with a description of Boxer, a reconstructible medium that we are designing for particular applications in education.

Boxer development is supported by the National Science Foundation under Grant MDR-85-96025 and Grant MDR-86-42177, and by the Defense Advanced Research Projects Agency of the Department of Defense, and monitored by the Office of Naval Research under Contract N00014-83-K-0125.

RECONSTRUCTIBLE MEDIA

Computers are commonly used for text processing, although most text written using computers is still intended to be ultimately printed on paper. As more and more people have access to computers, it becomes increasingly worthwhile to exploit the possibilities of the computer screen itself as an expressive medium. It is easy to imagine interactive books with elaborately structured text, moving illustrations, built-in simulations, and special-purpose dynamic tools whose interactive capabilities go far beyond canned presentations. For instance, a science textbook on optics could include text with multiple organizations and multiple means of access; moving illustrations of the wave properties of light; simulations that perform ray tracing through lenses and mirrors; databases of optical properties of various substances; and graphing and analysis tools for processing experimental data gathered through photoelectric sensors.

A computer-based medium for constructing such a book may seem like a great advance over present-day printed media. Yet, if viewed only as a way to produce fancy books, it lacks an essential quality necessary for a truly popular medium: the possibility for personal construction by users at all levels of competence. The optics book represents the "grand image" of the medium, such as one would see in commercially produced textbooks or in a finished novel. A popular medium of expression, however, must also be usable in ways that might suit the personal needs of children, teachers, or other noncomputer specialists.

In particular, a popular computational medium must be easy to program. In the hypothetical optics book, for example, all elements of the book, beyond simply entering the text, would be created through some sort of programming. We would like everyone to have access to the same kinds of tools used for constructing the book. The medium should also serve beginners and casual users, even if they never reach the stage of producing an exemplar of the grand image.

One major benefit of programmability is that even professionally produced items become changeable, adaptable, fragmentable, and quotable in ways that present software is not. Not only would professionals be able to construct grand images, but others would be able to *reconstruct* personalized versions of these same images. Giving all users access to the behind-the-scenes organization of an interactive book means that new ideas—the dynamic equivalent of famous quotations—can enter the culture with the medium.

Beyond interactive books, a reconstructible medium should allow people to build personalized computational tools and easily modify tools they have gotten from others. This concept is in strong contrast to the current situation in applications software—professionals are designing tools only for large populations with a common need. Since only experts can craft such systems or tune them to particular purposes, designers must predict every possible variation that users might need, and supply often ad hoc methods of selecting among options. With a reconstructible medium, there is no need to play guessing games to this extent, and changes to any application tool can be made uniformly—through programming.

PROGRAMMING LANGUAGES REVISITED

Most research into the design of programming languages has been in the tradition of programming as a way for experts to get computers to perform complex tasks efficiently and reliably. Programming as a means of controlling an interactive medium sets very different needs and constraints on languages. Here are some traditional desiderata for programming languages that ultimately are not of major importance in creating a popular medium:

- Formal simplicity—a computer scientist's or a mathematician's measures of simplicity are simply not at issue. A better criterion is accessibility to a seven-year-old child.
- Efficiency—there are so many other important attributes of a language for controlling an interactive medium that efficiency must take a backseat. If present-generation machines are not powerful enough, we can wait for the next.
- Verifiability—rigor is not a primary requirement of an expressive medium. Much more important is the ability to accommodate a wide variety of expressive styles.
- Uniformity—the demands of multifunctionality are great, and it is likely that some degree of uniformity will have to be sacrificed.

More telling is what emerges as important to a broadly based computational medium:

- Understandability—this is a primary and unavoidable goal if nonexperts are to successfully use this new medium.
- Tuned toward common, directly useful functionalities—if the medium is to be useful and widely used, it will have to seem more "familiar"; for example, basing data structures on text and pictures rather than on abstract, though perhaps more general objects such as arrays or lists.
- Tuned toward small tasks—the ability to implement simple ideas easily is much more important in this context than the ability to do complex tasks efficiently.
- Interaction—user interfaces are often considered to be separable from programming language semantics and almost an afterthought in language design. Worse, most present languages assume only character-stream input and output. A useful medium must be much more flexibly interactive.

BOXER

Much of Boxer's character is determined by two key principles—the *spatial metaphor* and *naive realism*.

People have a great deal of commonsense knowledge about space that can be used to make computers more comprehensible. The spatial metaphor encourages people to interpret the organization of the computational system in terms of spatial relationships.[1] Using a Boxer system is like moving around in a large two-dimensional space. All computational objects are represented in terms of boxes, which are regions on the screen that contain text, graphics, or other boxes. Boxes within boxes represent hierarchical structures. For example, (1) a variable is a box containing the variable's value; for a compound data structure (such as a record with named fields) the variable contains other variables; (2) a program is a box containing the program text; internal subprocedures and variables (as in block-structured programs) are represented as subboxes. When you enter a box (by moving the cursor into it),

[1] Boxer's use of the spatial metaphor was encouraged by work on spatial data management systems at the MIT Department of Architecture [2].

you gain access to its contents. Thus, any box can be a special-purpose environment with its own data and behavior (programs).

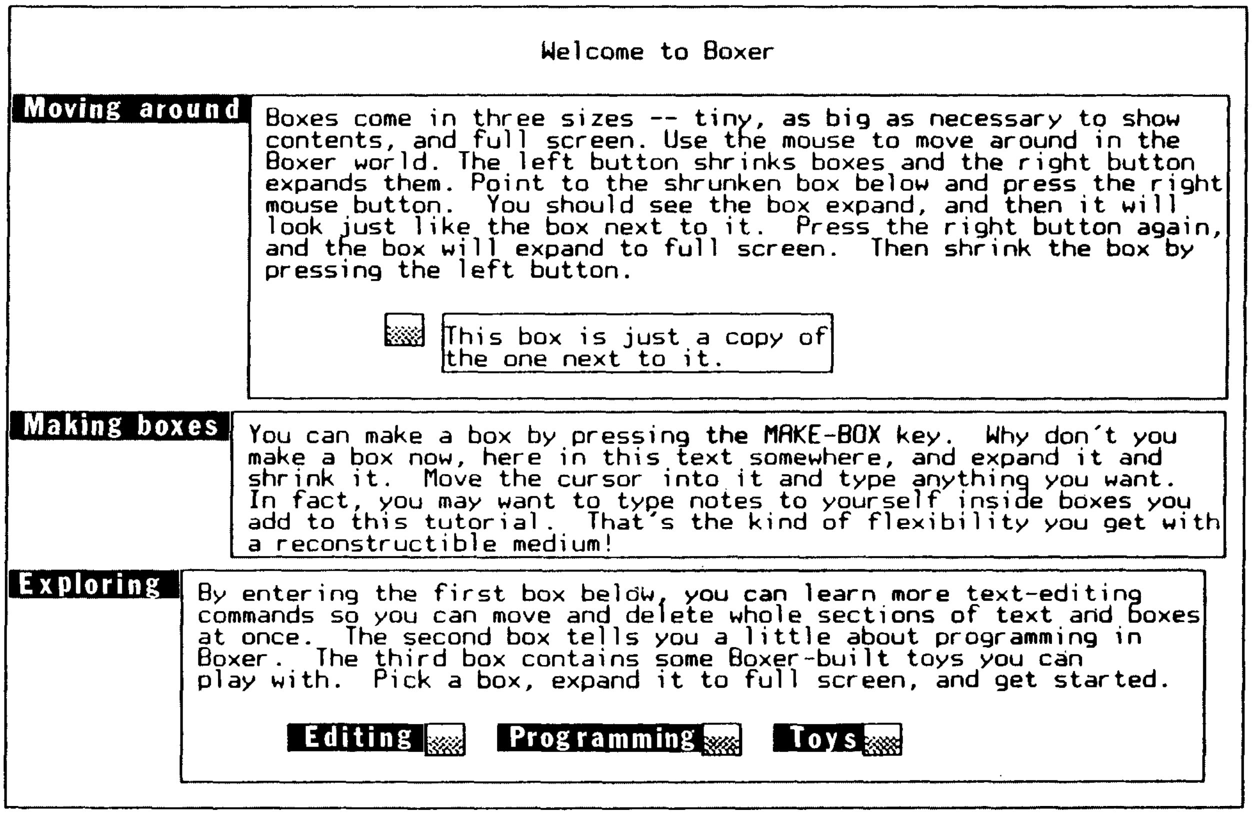
Naive realism is an extension of the "what you see is what you have" idea that has become commonplace in the design of text editors and spreadsheets, but not for programming languages. The point is that users should be able to pretend that what they see on the screen is their computational world in its entirety. For example, (1) any text that appears on the screen—whether typed by the system, entered by the user, or constructed by a program—can be moved, copied, modified, or (if it is program text) evaluated; (2) you can change the value of a variable simply by altering the contents of the variable box on the screen. If a program modifies the value of a variable, the contents of the box will be automatically updated on the screen. In general, there is no need to query the system to display its state, nor any need to invoke a state-change operation to affect the system indirectly.

Following are some highlights of Boxer's most important features. Our aim is to suggest how users can move smoothly from simple text and data manipulation, through modifying and producing personal computational tools, to dealing with larger systems such as interactive books. Of key interest is the way in which the basic box structure is elaborated to support many important functions of a reconstructible medium. (Additional examples and discussion of the theoretical and empirical motivations for specific choices in Boxer's design can be found in [3] and [4].)

Boxer currently exists as a prototype, including all of the features described below, implemented on Symbolics and Texas Instruments Lisp machines. We are about to start implementation on a more modest machine so that we can begin testing the system extensively in a variety of settings.

Boxes and Text

The text shown in Figure 1 is part of a tutorial on Boxer, organized using boxes. Note particularly the way in which, by shrinking and expanding boxes,



Welcome to Boxer

Moving around

Boxes come in three sizes -- tiny, as big as necessary to show contents, and full screen. Use the mouse to move around in the Boxer world. The left button shrinks boxes and the right button expands them. Point to the shrunken box below and press the right mouse button. You should see the box expand, and then it will look just like the box next to it. Press the right button again, and the box will expand to full screen. Then shrink the box by pressing the left button.

This box is just a copy of the one next to it.

Making boxes

You can make a box by pressing the MAKE-BOX key. Why don't you make a box now, here in this text somewhere, and expand it and shrink it. Move the cursor into it and type anything you want. In fact, you may want to type notes to yourself inside boxes you add to this tutorial. That's the kind of flexibility you get with a reconstructible medium!

Exploring

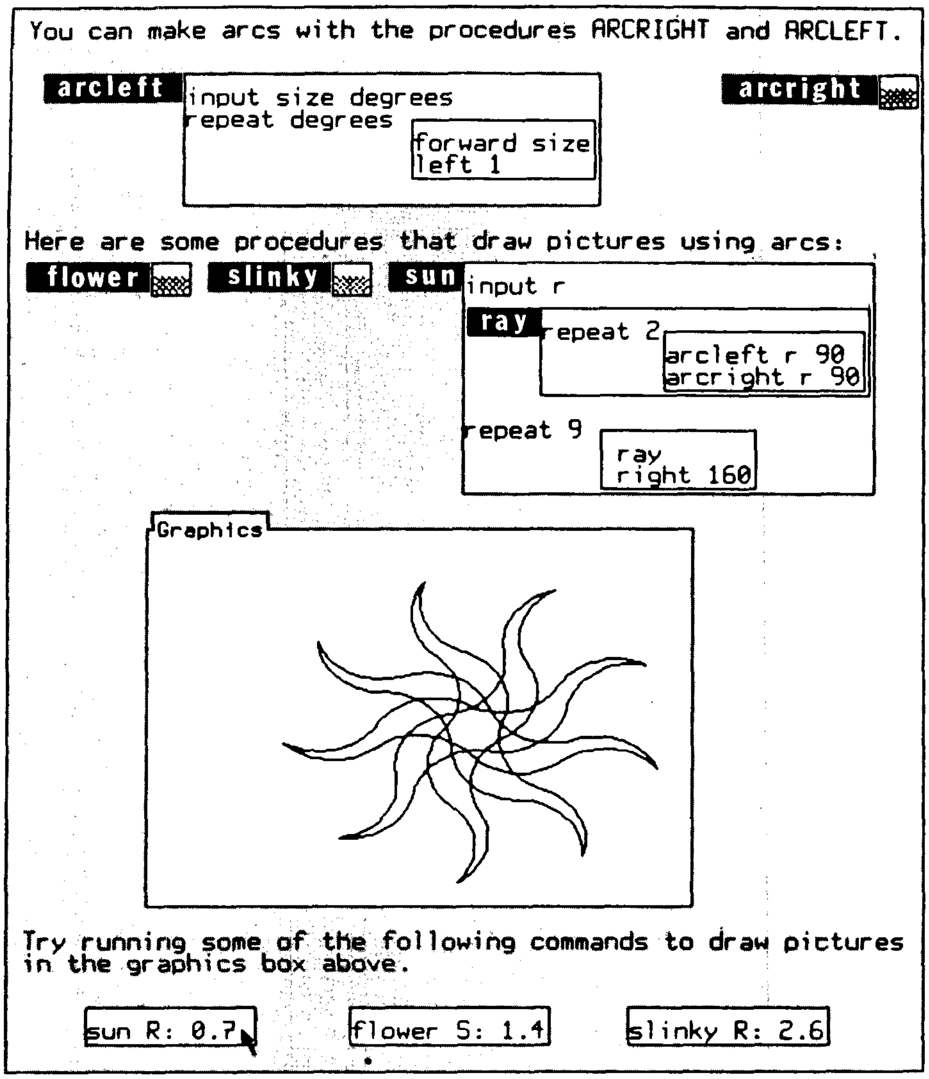
By entering the first box below, you can learn more text-editing commands so you can move and delete whole sections of text and boxes at once. The second box tells you a little about programming in Boxer. The third box contains some Boxer-built toys you can play with. Pick a box, expand it to full screen, and get started.

A mouse is used as a pointing device to move around the system and to shrink and expand boxes.

FIGURE 1. A "Page" from a Boxer Tutorial

detail can be hidden or shown for brief inspection. By expanding a box to full screen, the user effectively enters a subenvironment (box). Figure 1 shows how essentially all of the mechanics of making, inspecting, and modifying boxes can be learned and used without knowing anything at all about programming.

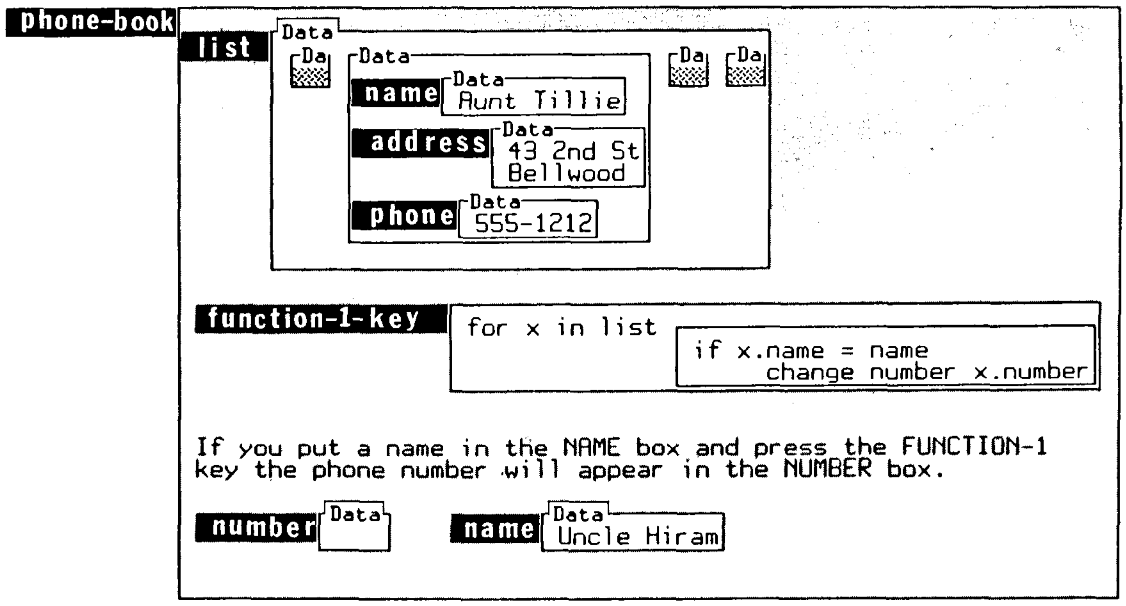
At first glance, boxes may seem to be nothing more than a variant on the windows used in many display-oriented systems. Windows, however, have no computational semantics except as places to display interaction with a program or application—a window's position on the screen and its relation to other windows do not generally reflect any information about the objects in the computational system. Boxes, in contrast, *are* the system's computational



```
You can make arcs with the procedures ARCRIGHT and ARCLEFT.

arcleft  input size degrees                          arcright
         repeat degrees
                         forward size
                         left 1

Here are some procedures that draw pictures using arcs:
flower   slinky   sun  input r
                       ray  repeat 2
                                      arcleft r 90
                                      arcright r 90

                       repeat 9
                                 ray
                                 right 160

Graphics

Try running some of the following commands to draw pictures
in the graphics box above.

   sun R: 0.7        flower S: 1.4        slinky R: 2.6
```

Graphics appear inside regions of the screen called graphics boxes. The box labeled SUN defines the procedure that drew the design in the graphics box above. Definitions (boxes with name tabs on them) can be invoked anywhere within the box in which they appear. This scoping rule provides a general block-structuring capability that has been used here to include RAY as an internal subprocedure of SUN. The bottom line of the screen is a menu from which the user can select commands to be run. Here, as indicated by the small arrow (mouse cursor), the user has selected SUN with an input of 0.7. The letter R followed by a colon is a prompt generated by the system to indicate that the SUN procedure requires one argument, named R. Such prompts are generated automatically whenever users type the name of a defined procedure and press a help key. In general, any text on the screen can be modified or run.

FIGURE 2. Turtle Graphics in Boxer



```
for x in list
    if x.name = name
       change number x.number
```

If you put a name in the NAME box and press the FUNCTION-1 key the phone number will appear in the NUMBER box.

This box is a utility for finding phone numbers. Its local database is a box called LIST that contains subboxes with a standard format. The procedure named FUNCTION-1-KEY will be run whenever the FUNCTION-1 key is pressed.

FIGURE 3. The PHONE-BOOK Box

objects, and box containment reflects meanings such as subprocedures as parts of procedures and records as parts of databases. The part-whole relation implied by box containment is fully operational, and the box structure can be traversed, inspected, and changed by programs as well as manually. Boxer's spatial metaphor is a recognition that spatial relations are extraordinarily expressive and should not be wasted by being used only for transient needs (where there is space to pop up a window) or by divorcing spatial relations from fundamental semantics. Similarly, the concept of naive realism dictates that one should see computational objects, not just interfaces to them.

Simple Programming

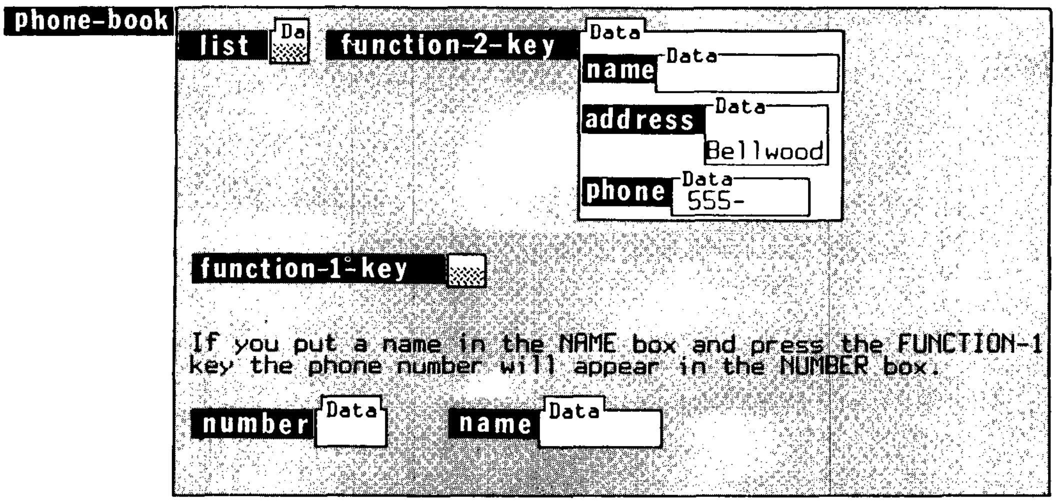
Figure 2 shows some simple Boxer programs that draw designs using the turtle graphics commands introduced in the educational computer language Logo.[2] It is important to observe how text, programs, and graphics have been intermixed to produce a tutorial about drawing using arcs; note in particular that the bottom line of the figure is a menu of commands for the user to try. This menu was created simply by typing the commands in place and leaving them on the screen to be run or modified—the principle of naive realism dictates that anything that appears on the screen should be manipulable in this way. Here, as indicated by the mouse cursor, the user has selected to run the command SUN with an input of 0.7. The graphics output appears in a graphics box. Graphics boxes can also be named, copied, and moved using Boxer's editor interface.

Two of the procedure boxes in Figure 2, ARCLEFT and SUN, have been expanded to illustrate the form of Boxer procedures. Note that SUN contains an internal procedure RAY; box containment is being used here to implement block structuring of procedures. In general, the scoping rules of Boxer—definitions are accessible inside a box, but not outside of it—allow for boxes to be used as environments that users enter to gain access to procedures and data defined inside. Assimilating the computational idea of scoping into the intuitive notion of "inside" is central to the spatial metaphor.

A Simple Database Tool

The box named PHONE-BOOK, shown in Figure 3, is a simple utility for storing and retrieving phone numbers. When a user types a name in the box marked NAME and presses the FUNCTION-1 key, the corresponding phone number will appear in the box marked NUMBER. PHONE-BOOK contains a database called LIST, which is a box containing other boxes with a standard format: subboxes NAME, ADDRESS,

[2] Boxer is in many ways an extension of Logo. Simple Boxer procedures—especially those for graphics—resemble Logo procedures. Our work in Boxer has been motivated largely by a desire to extend Logo-style activities to a much broader range of possibilities. Although Logo took very seriously the benefits of making programming a popular activity, it did not begin with the image of a medium encompassing written language, hierarchical structures, databases, and interactive graphical tools. (See [6] for an overview of Logo, and [1] for a development of geometry based on turtle graphics.)



If you put a name in the NAME box and press the FUNCTION-1 key the phone number will appear in the NUMBER box.

When the FUNCTION-2 key is pressed, Boxer will insert at the cursor a template for a record with address and phone fields partially filled in that can be added to the LIST. In order to make this extension, the user needed only to type the template and label it with the name FUNCTION-2-KEY. (The template shown here is especially useful if you happen to know many people who live in Bellwood whose phone numbers begin with 555.)

FIGURE 4. Extension to the PHONE-BOOK

and PHONE. The boxes in the database, as well as the NAME and NUMBER boxes, are *data boxes.* Marking a box as data indicates that, when the box is used by a program, the contents are to be interpreted as literal text rather than as a program to be executed. (This is similar to the use of QUOTE in Lisp.) Named data boxes are variables.

The FUNCTION-1-KEY is a procedure that looks up the designated name and supplies the corresponding phone number.[3] Boxer contains pattern-matching capabilities that make the lookup operation trivial, but we have written the procedure here in a way that illustrates some more basic Boxer capabilities: A FOR loop steps through the LIST, searching for a box whose NAME field matches the designated NAME. When the box is found, the NUMBER box is changed to the corresponding PHONE field. (There are obvious improvements to be made here, such as stopping the iteration when a match is found, and using an ordered database, but we have chosen to show only the very simplest procedure.) Note the use of the dot syntax <box-name> . <subbox-name> for specifying named subboxes of a box.

Boxer's naive realism automatically supplies an input/output mechanism for the procedure. The boxes NAME and NUMBER are ordinary variables. Typing something in the NAME box automatically sets the variable NAME that is referenced by the procedure. Similarly, when the procedure changes the variable NUMBER, the new contents will automatically appear on the screen in the NUMBER box.

In a user's overall Boxer environment, PHONE-BOOK is a special-purpose subenvironment. Boxer's scoping rules dictate that the binding of the function key to the lookup procedure will be active only when the user enters the PHONE-BOOK box. Other boxes in the system are free to bind this function key (or any other key) for their own purposes.

Extending the Database Tool

It is easy to make personalized extensions to the PHONE-BOOK as illustrated in Figure 4. The point of the phone-book example is not that everyone should write a phone-number-fetching procedure from scratch; rather, we want to illustrate how Boxer enables people to build or modify their own little tools of whatever idiosyncratic sort. Boxer is an environment designed for invention and functionality in little pieces, where understanding the system better in any particular context results in more power generally. For example, learning how to change the value of a variable is far more than a trick for the PHONE-BOOK program; it is an essential feature that allows users to modify any piece of Boxer.

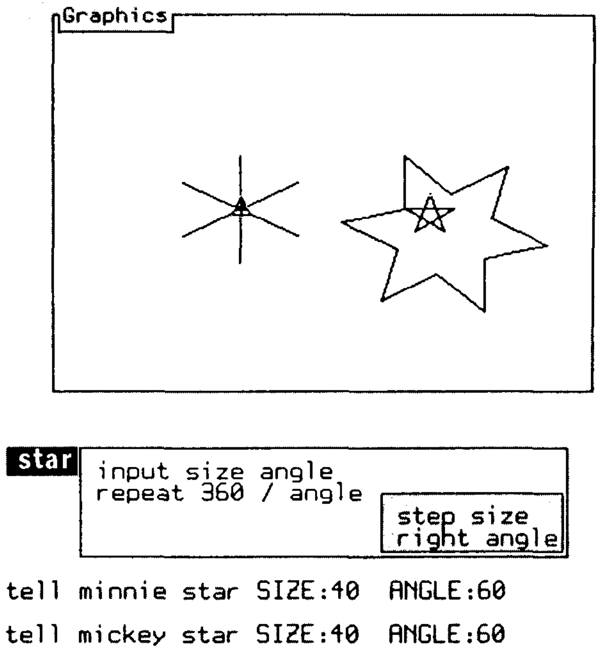
[3] The use of -KEY as a suffix automatically binds the operation to the specified key. Any key can be bound in this way.

Objects and Message Passing

The graphics box in Figure 5a is the home of two graphical objects (known in Boxer as *sprites*) named Minnie and Mickey. Also shown is the STAR procedure, which takes two parameters, SIZE and ANGLE, and draws a symmetric shape by repeating 360/ANGLE times the following sequence: Call the procedure STEP with an input of SIZE, and then the procedure RIGHT with an input of ANGLE. In the figure, Minnie and Mickey have each been told to perform STAR with a SIZE of 40 and an ANGLE of 60. Observe that the two sprites draw different designs. This is because, as we shall see below, Minnie and Mickey use different STEP procedures when executing STAR.

Turtle graphics, and the sprite extensions that appear in some versions of Logo, are known to be congenial forms of interaction for simple graphics programming. But the fact that the state of these graphical objects cannot be seen and directly manipulated violates the naive realism principle. Boxer is therefore arranged so that a graphics box is only an alternate form for an ordinary box structure that includes sprites as subboxes. Attributes of sprites—their position, heading, speed (if in motion), and shape (expressed as the procedure that draws the shape) are visible and manipulable as ordinary variables. Changes to these variables, whether by direct editing or under program control, automatically affect the graphical representation.

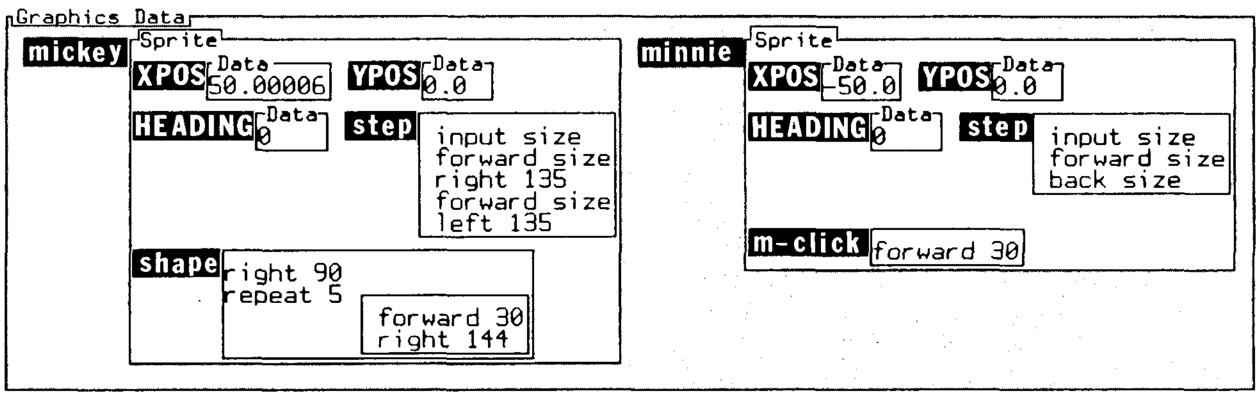
Figure 5b shows the nongraphical (data) version of the same graphics box with its resident sprites Minnie and Mickey. Minnie and Mickey each have their own version of the procedure STEP, which



```
star
input size angle
repeat 360 / angle
    step size
    right angle

tell minnie star SIZE:40  ANGLE:60
tell mickey star SIZE:40  ANGLE:60
```

This graphics box contains two sprites—Minnie (the small triangle) and Mickey (the small five-pointed star). Each sprite has been told to run the procedure called STAR. The designs drawn are different, because each sprite has its own definition of the STEP procedure when executing STAR.

FIGURE 5a. A Graphics Box



```
Graphics Data
mickey  Sprite                                  minnie  Sprite
        XPOS Data 50.00006   YPOS Data 0.0              XPOS Data -50.0   YPOS Data 0.0
        HEADING Data 0       step                       HEADING Data 0    step
                               input size                                   input size
                               forward size                                 forward size
                               right 135                                    back size
                               forward size
                               left 135                                 m-click forward 30
        shape right 90
              repeat 5
                   forward 30
                   right 144
```

Here is the graphics box of Figure 5a, shown in data form so that all of its computational structure is visible, changeable, and extendable in ordinary Boxer textual format. The XPOS, YPOS, and HEADING variables show the position and heading of each sprite. In addition, Mickey has a SHAPE procedure that makes him appear as a five-pointed star; Minnie has no SHAPE procedure and hence appears in the default shape, a triangle.

FIGURE 5b. The Data Form of a Graphics Box

they use when they perform STAR. Naturally, additional local procedures or variables can be added at any time simply by entering the sprite box and typing the definition in place.

Packaging local data and procedures together and organizing computations by sending messages to these packages via TELL are paradigms of *object-oriented programming*, popularized by the language Smalltalk [5]. Boxer's spatial metaphor assimilates such packaging to box containment, thereby making the object structure visible and concrete. Object-oriented programming in Boxer is not limited to sprites and graphics. Any box can be told remotely to execute any command that might ordinarily be locally executed from within that box.

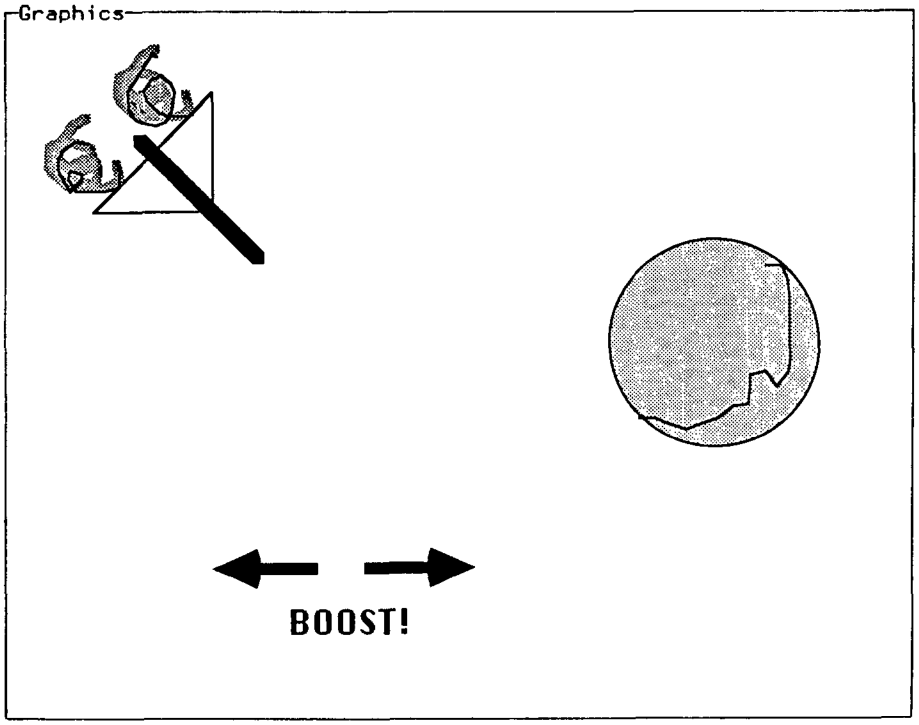
Point-and-Poke Interaction

Figure 5b also reveals that Minnie contains a procedure named M-CLICK. A sprite's M-CLICK procedure is automatically executed whenever the middle mouse button is clicked over the sprite in graphics presentation. In this case, pointing at Minnie with the mouse and clicking the button will make her go forward.

A simple game that could be constructed by a child using a few lines of code and the above interactive capabilities is shown in Figure 6. The graphics box contains five sprites: a planet, a rocket, two arrows, and a BOOST! icon. Because sprites have touch-sensing capabilities, it is easy to include obstacles, such as the planet, that the rocket must avoid in order not to crash.

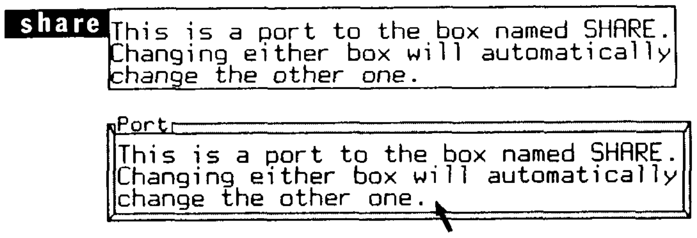
Ports and Sharing

Boxer's spatial metaphor should prove to be an important factor in helping people deal with computational structures. It does, however, impose a significant constraint on the structures that can be represented—box containment is a strictly hierarchical relation. Using containment only, shared data structures could not be represented, nor could two widely separated boxes be viewed at the same time without moving one of them and thus changing the state of the system.



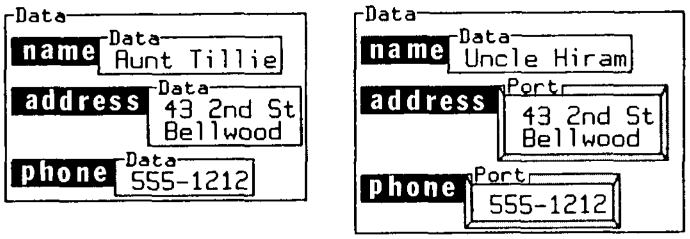
A graphics box with five sprites includes a control panel for piloting a rocket. A click on the BOOST! sprite gives the rocket a little thrust; a click on the arrows rotates the rocket to the right or left.

FIGURE 6. Interaction with Sprites



share
This is a port to the box named SHARE.
Changing either box will automatically
change the other one.

Port
This is a port to the box named SHARE.
Changing either box will automatically
change the other one.



Data
name Data Aunt Tillie
address Data 43 2nd St Bellwood
phone Data 555-1212

Data
name Data Uncle Hiram
address Port 43 2nd St Bellwood
phone Port 555-1212

The second box is a port to (the alternate view of) the box named SHARE above it. Text typed into the port at the arrow cursor has also automatically appeared in the SHARE box.

The two lower boxes illustrate how ports can be used to implement shared data: Any change to the ADDRESS or PHONE fields in either box will also appear in the other box.

FIGURE 7. Examples of Port Structure in Boxer

In order to overcome this limitation, Boxer includes a structure called a *port*, which is simply a view of a box at some other place in the system. A port behaves in most respects identically to the box it views—any change in one will automatically cause the same change in the other. Figure 7 shows a typical application of ports. Ports can also be used to provide cross-referencing in interactive books or databases, to obtain multiple views of computational objects (e.g., to view a sprite in its graphics and data form at the same time), to share combinations of local data and procedure among objects, and to implement various nonstandard scoping disciplines for procedures.

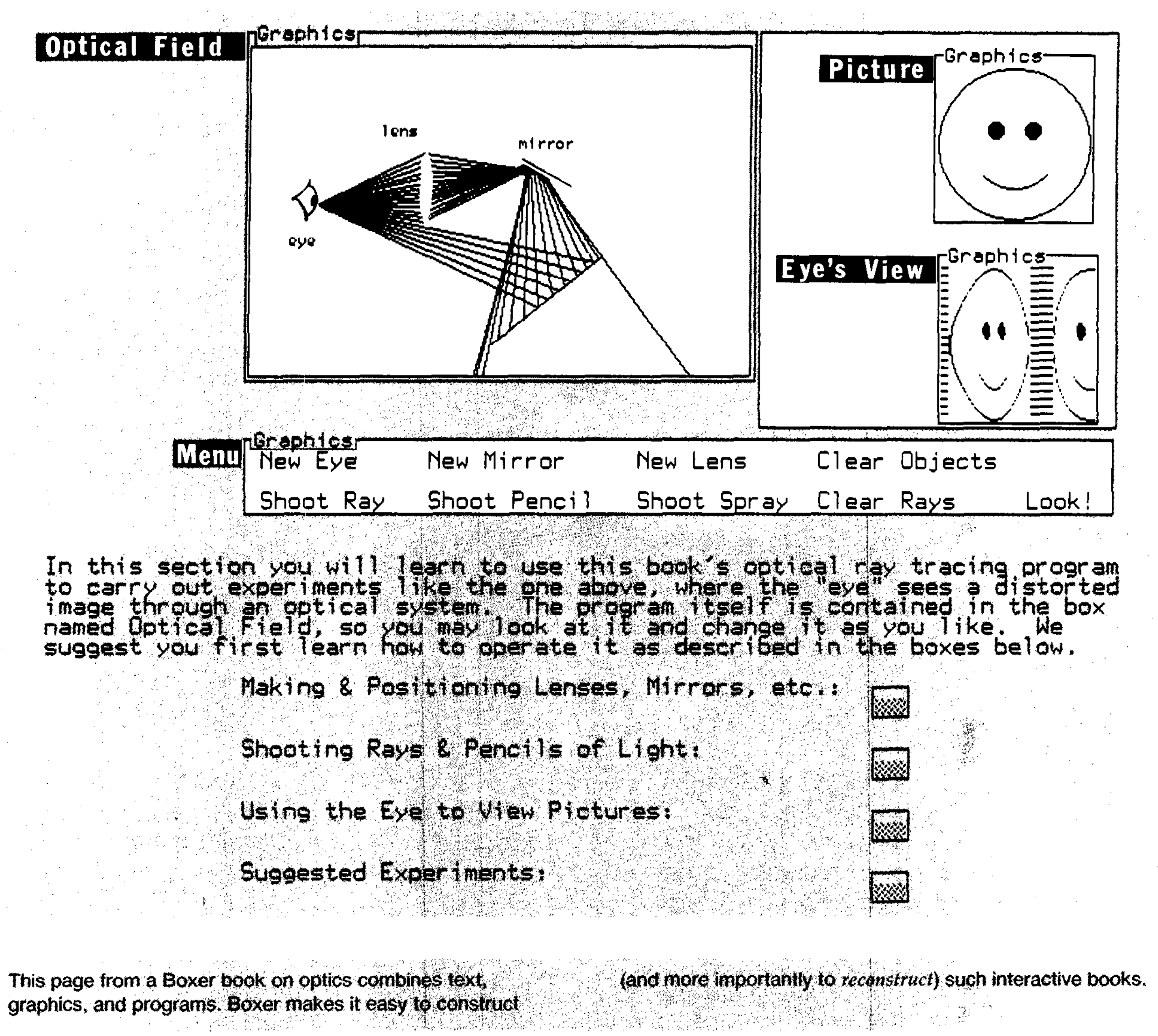
THE GRAND IMAGE

As a final example, Figure 8, page 868, shows a page from the interactive book on optics proposed earlier. It should be clear how such a book can be constructed in Boxer as a natural evolution of text editing combined with writing simple procedures. It should also be apparent that, once produced, such a book can be readily modified and *reconstructed* by its users—both in small ways, as by adding marginal notes, and more extensively, as by modifying the simulations and tools included in the book.

It is exciting to contemplate the possibilities of interactive books, published on high-density media, such as compact disks that would appear to users as parts of their personally changeable Boxer systems. We would also love to extend Boxer to incorporate sound and high-quality moving pictures in graphics boxes. This is, however, the grand image of the medium, and most users' time will be spent in far more modest pursuits than constructing such books. The important point is that we can imagine a progression from the equivalent of scribbling in this new medium, as children scribble with pencil and paper, to the profound "scribblings" of experts—a path in which each new stage of understanding and competence is rewarded with new opportunities for personal expression.

CONCLUSION

Boxer challenges, in a small way, the current view of programming languages. More significantly, it challenges the current view of what programming might be like, and for whom and for what purposes programming languages should be created. We have argued that some computer languages should be designed for laypeople, and have presented an image of how computation could be used as the basis for a popular, expressive, and reconstructible medium. Computers will become substantially more powerful instruments of educational and social change to the extent that such an image can be realized.



In this section you will learn to use this book's optical ray tracing program to carry out experiments like the one above, where the "eye" sees a distorted image through an optical system. The program itself is contained in the box named Optical Field, so you may look at it and change it as you like. We suggest you first learn how to operate it as described in the boxes below.

Making & Positioning Lenses, Mirrors, etc.: ▨

Shooting Rays & Pencils of Light: ▨

Using the Eye to View Pictures: ▨

Suggested Experiments: ▨

This page from a Boxer book on optics combines text, graphics, and programs. Boxer makes it easy to construct (and more importantly to *reconstruct*) such interactive books.

FIGURE 8. A Sample Page from an Interactive Book

Acknowledgments. We gratefully acknowledge the efforts of all those members of the Boxer Groups at MIT and Berkeley who have helped to make Boxer (almost) a reality. Special thanks to Michael Eisenberg, Gregor Kiczales, Leigh Klotz, Ed Lay, and Jeremy Roschelle.

REFERENCES
1. Abelson, H., and diSessa, A.A. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics.* MIT Press, Cambridge, Mass., 1981.
2. Bolt, R.A. Spatial data-management. Rep., Dept. of Architecture, MIT, Cambridge, Mass., 1979.
3. diSessa, A.A. A principled design for an integrated computational environment. *Hum.-Comput. Interaction 1*, 1 (1985), 1–47.
4. diSessa, A.A. Notes on the future of programming: Breaking the utility barrier. In *User-Centered Systems Design*, D. Norman and S. Draper, Eds. Lawrence Erlbaum, Hillsdale, N.J., 1986.
5. Goldberg, A., and Robson, D. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley, Reading, Mass., 1983.
6. Papert, S. *Mindstorms: Computers, Children and Powerful Ideas.* Basic Books, New York, 1980.

CR Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; D.2.6 [**Software Engineering**]: Programming Environments—*Boxer*; D.3.0 [**Programming Languages**]: General—*Boxer*; H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; K.3.0 [**Computers and Education**]: General
General Terms: Human Factors, Languages
Additional Key Words and Phrases: computational media

Authors' Present Address: Andrea A. diSessa, School of Education, University of California, Berkeley, CA 94720; Harold Abelson, Laboratory for Computer Science, M.I.T., Cambridge, MA 02139.