COMPUTER-CONTROLLED DISPLAY DEMONSTRATIONS OF DYNAMIC CONCEPTS IN COMPUTER SCIENCE

David R. Levine
Technical Report #37

To be presented at:

Rutgers University
The State University of N. J.
Department of Computer Science
New Brunswick, New Jersey 08903
I. INTRODUCTION

The presentation of a new concept lying totally outside of the student's prior experience constitutes a major educational challenge. Somehow no amount of explanation, handwaving, whatever, can take the place of direct personal experience. Many instructors just ignore the problem and go right on*, leaving the students to pick up an understanding piecemeal on their own. A more conscientious instructor will attempt to provide the students with some kind of personal exposure to the new concept. In physics, for instance, the lecturer always has a table full of demonstration apparatus at hand. In sum, a picture can be worth a thousand words, and the real thing is even better. Imagine how you'd explain about theatrical drama to someone who'd never seen a play!

Computer Science contains a large number of new concepts well outside most students' prior experience. An important characteristic of many of these concepts is their dynamic nature: the execution of a program, assignment of a value to a variable, change of machine state following an interrupt, convergence on a root. By using the computer itself as my demonstration apparatus, I propose to give the students personal exposure to the concepts in action. With a low-cost interactive CRT terminal and video projector, I can show various aspects of a program executing, for instance, at comparatively low investment in machine and people time.

Personal exposure seems to be the basic key to deep understanding. To teach, we use all sorts of other techniques to reach the students:
- we describe endlessly, aided by blackboard, sometimes by pictures;
- we appeal to similar ideas in other areas, constructing analogies to experiences we hope the student has actually had.

*This may occur because once one has sufficient experience, the new concept is "obvious" and the instructor forgets that the students will have trouble.

(and hoping the analogies won't be too misleading...)
- we demonstrate the new concepts (e.g. a film class is shown movies, a physics class is shown demonstration apparatus);
- we provide actual contact with the new concept, through homework, laboratory and field trips.

Computer Science, like other subjects, is taught using these traditional methods, with reasonable success. The situation is far from optimal, however.
- description is often awkward and incomplete due to the dynamic nature of the concepts involved, and the static character of the blackboard. Also, the strict mechanistic nature of the subject demands a level of precision and accuracy not easily attained by a lecturer.
- an appeal to similar ideas is often very difficult, since computer science is quite different from most of the student's prior experience. The ubiquitous few examples which are workable (recipes, changing a tire) merely attest to the paucity of alternatives.
- demonstration of the concept is the subject of this paper. Direct demonstration is, of course, infeasible, since the actual information resides in electrical signals of extremely transient nature; not even access to an actual machine and its lights will be of much help.
- achieving actual contact is even a bit hard. The student's most direct contact is through the programs he writes and runs; but printout supplies him mainly with the results if having run a program, from which he must deduce what actually transpired. (e.g. Dijkstra, [1972], p.16)

2. THE DEMONSTRATIONS

To transform the action of the computer into something visible, I employ a Video Display Unit connected on-line to an interactive time-sharing system. Using a video projector, I have successfully employed program-driven demonstrations in teaching an introductory programming course at Rutgers. Some of my colleagues
have begun to use video demonstration in the machine-language course as well: Development is underway on other projects. The programs currently include:

- A PL/I system which shows a program in execution, one statement at a time. This demonstration system helps to show the concept of sequential execution, as well as displaying the execution of the various statement types and the operation of simple algorithms.

- A machine-language simulator: This project shows the basic instruction fetch, decode, and execute cycle of a simple computer (FACELT), plus the operation of the various instructions in its repertoire.

- Finite-State Machine: Illustrates the interpretation algorithm, showing the state of the machine, input and output actions, state transitions, and acceptance of a string.

- LISP: demonstrates the correspondence between s-expressions (textual form) and list structure (internal form). Also shows recursive operation of the evaluation algorithm.

- Root finding: graphically shows the action and convergence of various methods.

Examples of many of these projects, accompanied by more complete explanations, are given below.

I should, at this point, acknowledge the work of others. The PL/I system has been entirely my own doing. Many of its features resemble projects reported by Shapiro & Witmer (1974), though development was independent. The PLATO project also has developed an interactive PL/I system for educational purposes (Wilcox et al., 1975), which is display-oriented and quite powerful. To my knowledge, their system does not specifically include the ability to demonstrate the step-by-step execution of PL/I.

The other projects were done as course work for a graduate seminar at Rutgers. (A complete list of these projects is given in the appendix). The machine-language simulator is similar to a project reported by Shapiro (1974), and to another reported by Schwepp (1973) on and for a Datapoint 2200. All are antedated, however, by an elaborate KRT machine-language debugging program called RAID, written by Paul Stygar for the Stanford PDP-1 system in 1965. (McCarthy et al., 1967).

Also Trzes (1974) describes an operating system simulation with accompanying display. Baucker (1975) has done several dynamic demonstrations, ranging from the operation of a sort program to a mini-LOGO interpreter; these programs, however, generated films rather than working on-line.

3. BASIC STRUCTURE
3.1 Description:

The computer-driven demonstrations possess a two-level structure. At the base

lies an algorithm capable of actually performing the task at hand -- be that finding a root, obeying machine instructions, or executing PL/I. To this basic algorithm has been added a second level, which serves to display the status of the main algorithm on a video display unit. When properly done, the demonstration level is unobtrusive, and the user is aware mainly of the primary algorithm. In particular, there is (potentially) access to the full flexibility of that algorithm in specifying data at execution time.

A contrasting situation is presented by a film or videotape demonstration. In these media, the entire emphasis is on the production of a "correct" visual image using whatever editing tools may be available. During presentation one has, moreover, no control over the course of the "algorithm" being demonstrated -- if indeed there is an algorithm present at all. The computer itself can be guilty of this presentation-only approach. Some of the programs I constructed to do my early demonstrations began by printing a great deal of literal text -- the "program" being demonstrated. The actual execution was in fact algorithmic, but the correspondence between the algorithm being executed and the text of the program on the screen was purely due to the grace of the author.

By analogy, consider the problem of demonstrating the locomotion of a tiny amoeba to a large class. One might place a live animal under a microscope and attempt to project the image on a large screen for direct viewing. Alternatively, one could show a film of an amoeba. The latter approach would doubtless be available, brighter, clearer, and more interesting behavior, as the film has doubtless been carefully prepared, photographed with the best equipment at the right times, and edited for maximum clarity. Use of the projecting microscope corresponds to my two-level scheme: the teacher observes a real amoeba moving about as real amoebas do. If some chemical stimulus is appropriate, it can be administered as desired and its effects observed as they occur.

3.2 Flexibility and Credibility:

From the two-level scheme, we get two significant benefits: flexibility (for the teacher); and credibility (for the students).

Flexibility arises from the presence of the algorithm as the basis of the demonstration program. Presenting a demonstration involves executing the algorithm. The operator has full free to supply the program with whatever data it desires, thus tailoring the demonstration to his own needs.

Exactly what degree of flexibility exists will depend on the specific demonstration. The PL/I and machine-language demonstrations are extreme examples, as the "data" for the basic algorithm consist
of entire programs, thus affording almost complete freedom.

Credibility depends on being able to convince the student that the demonstration really reflects the algorithmic concept as claimed. The exercise of the flexibility component serves as proof of the existence of the basic algorithm, since the students can observe the program computing with unanticipated data right before their eyes. Without the flexibility in evidence, one could speculate that what appears on the screen has resulted merely from a clever sequence of text-output commands.

When the action of the demonstration depends on actual data entered then and there, no "simple explanation" suffices: the demonstration must be honestly algorithmic.

I should like to emphasize the importance of the credibility issue. We are, as teachers, asking the student to explore a new concept, often one which he finds strange and confusing. As part of learning a concept, one must internalize definitions of what the concept is and what it is not. If the student is unsure of the accuracy of the demonstration, he will be unable to make full use of the information therein for molding his understanding of the concept. In particular, if there is some doubt about the credibility of the demonstration, then a point which the student finds troublesome can easily be dismissed as merely an artifact of the demonstration.

3.3 Ramifications:

Use of a real-time, two-level scheme has certain ramifications.

1. One needs to develop both the algorithm itself and the display routines--often a non-trivial task. For example, the PL/1 demonstrator system contains a full compiler-interpreter for a PL/1 subset. Because the program will operate in real time, any difficulties must be fixed; they cannot be edited out or corrected manually.

2. A moderately sophisticated computer system must be available to carry out the algorithm, support the development work, and carry out the demonstration; reliability considerations may prove very troublesome. Furthermore, portability requirements restrict the choice of terminal type considerably.

For most of the work described herein, I have used a Datamedia Elite 2500 terminal driven by a DECsystem-10. Programming techniques make this possibility especially strong as they remove the time constraints from the production of the demonstrated image; one can easily make up a screen-full, photograph it, change it somehow (even manually), and continue at leisure.

ning was done in SIMULA-67. The equipment proved quite adequate at this stage of development.

3. The cost of an individual demonstration is fairly low, being merely the charge for running the program once, plus display overhead. (I assume that the algorithms being shown will generally involve simple computations.) This remains true, even though each demonstration may well differ from any previous one in the actual data used.

A further word on flexibility. It is not my aim to provide a new curriculum for other instructors; that approach leads nowhere. (C.f. PLATO.) Rather, I wish to provide a general-purpose tool, these demonstration systems, to help each with his own curriculum. At this price of learning to use the tool, these systems offer endless variety. The alternative technology, film or videotape, must fail on this count. Availability of a sufficient number of demonstration films could provide most of the necessary flexibility but impose a tremendous burden on the instructor who would first have to preview the entire selection. Unlike learning a new skill--how to use the demon program--previewing does not generalize to make the nth occurrence easier than the nth.

4. CONCEPTS

The dynamic display provides a new teaching tool, with new capabilities and restrictions. Although it has wide applicability one must still consider carefully just where it will be most useful; and then, exactly how the new tool should be employed.

To begin, one must identify and delineate the specific concept to be shown. Next, one must decide which examples best show off the concept; and then what points should be emphasized and at which level of detail. It will often prove necessary to rethink the teaching procedure in order to make the best use of the capabilities of the new tool, and at the same time to fit within its restrictions. On the latter point, I find that the limitations of the video display unit, though often vexing, provide a valuable impetus towards both clarity and brevity of presentation.

For me, the task of creating dynamic demonstrations provides interesting insights into the nature of perception and learning. I find that to be effective, one must show not only the change, but also its context. (One might even venture that change without context lacks interest just as physical motion is meaningless only as it is relative to some other object.) To learn a dynamic concept, one must be able to understand both the primitive action and its relation to the rest of the universe. How does it interact? How do

its effects synchronize with other events? Full comprehension involves the answers to these and other questions.

I feel that the comprehension of a complex process is acquired on various levels. One must come to understand the specific action of each constituent subprocess; grasp the interaction between each constituent, and perceive the larger picture, how the whole is created from its parts.

There is no particular reason to believe that all students approach this problem in the same way. One student might do best by first learning the detailed operation of each subprocess, and then move to consider their interactions. Another might prefer to fix his attention at a critical interaction and explore the other actions from that vantage point.

Within the demonstration, the sequence of actions is fixed by the operation of the underlying algorithm. The demonstration program will call attention to change as it occurs, thus providing a modicum of guidance; but the student is otherwise left to shift his attention to different areas of the display. This freedom also extends to the lecturer, who is expected to accompany the demonstration with a verbal explanation.

I try for a level of detail great enough to show the subprocess action and yet coarse enough to allow display of sufficient context. The implementation of this median policy requires that a great deal of clarity and simplification go into the crafting of a demonstration. One may find that the range of detail possible in a single demonstration is not sufficiently broad to encompass all of the material one has in mind. In this situation, it will be necessary to separate the concepts (e.g., arithmetic operator precedence and control of looping) and make two demonstrations, each with more manageable scope.

For example, the PL/I demonstration does not show detailed evaluation of arithmetic expressions. Operator precedence rules are merely minutiae at the level of an entire program, and distract from the comprehension of concepts like repetitive execution.

5. IMPLEMENTATION

An effective computer-controlled demonstration must be molded to the computer science concepts involved. Guided by that requirement, one must blend a theory of learning, such as discussed above, with the general technique of visual presentation.

5.1. Presentation:

One must decide what to display, and how to show it. The screen must contain enough information to establish an appropriate context, as well as containing various dynamic data. However, putting too much information on the display is likely to be confusing. Proper structuring and labeling are very important.

The display in the PL/I demonstration consists mainly of the program being demonstrated. The major design decision involved was to display the current value of the variables. One could show new values in the margin next to the appropriate assignment statements. I chose, rather, to show the variables in the upper right corner, in a permanent block. This has the effect of grouping together physically items that belong together logically. Individual variables are therefore easier to locate on the screen; or one can ignore, as a group, detailed value changes.

The finite-state machine demonstration (Garrison in CSS70) provided a much more difficult design task. The machine itself can be shown either as a state table or as a collection of arcs and nodes; both ways have advantages. The nature of the problem does not allow simple groupings as the operation of the machine involves a complex web of interactions. Each input symbol must be referred to the current state, which determines where to go next, thus leading to both output and a new state.

The graphical root-finder (Gwalthney in CSS70) had a severe context problem. As a root is approached it is necessary to change the scale of the display to observe the progressively smaller changes. Since the display medium is storage tube in this instance, zooming in was not possible, yet the old context is lost when the graph is redrawn on a new scale. An effective solution has been to include a reduced version of the old image in a corner of the new.

5.2. Attention:

The observer's attention must be directed to the proper area of the display, rapidly and smoothly, without destroying or distorting the information's unity, from which the context is established.

I have found certain methods that work. I use a blinking datum whenever a value changes, to attract attention to the new value (as in the PL/I demo.) To me, the blink has come to signify a new value, the arrival of which has been arrested, like a bird just landing. When an old, unchanged value blinks for attention, I find it disturbing. The FACET machine (Abrams in CSS70), for instance, blinks the accumulator contents at one phase of executing a conditional branch instruction. I feel uneasy about the constancy of a value I'm watching when it suddenly begins to blink--has it been changed when I didn't expect it? However, I also use the blinking feature (though not with datum) in the PL/I demonstration to show which path of an IF-THEN-ELSE has been chosen, by blinking the appropriate keyword.

Pointing at something (a cursor or arrow) draws attention to it. An arrow with a blinking component stands out. The arrow works well in the PL/I demonstration,
to identify the current statement; in other projects, it is less effective perhaps because it gets lost in the various shapes on the screen. The cursor seems best when consistently associated with just one facet of the demonstration. The PACET machine uses a cursor for the current instruction, effective address, and to highlight the accumulator during an arithmetic instruction; the multiplicity of use seems to lessen the overall effectiveness.

One project has found a particularly effective way to focus on change about to occur. (Interrupt demo, O'Reilly in CS 570). Prior to transmitting data from one point to another, the program "grows" a line along the path to be taken. (The line is erased after the data transmission occurs). Motion involved in creating the line catches one's attention: the presence of the line where none existed before maintains the focus for attention even after the motion has passed.

Color would appear to be an excellent technique of directing attention, but is not available to us. Some success has come from the use of different levels of intensity, though the contrast is not too great.

All attention-catching techniques must be used sparingly. Not only are the attention-grabbers distracting, but they are effective only because they contrast with the other material of the display. Two similar cursors, for example, seem about all the screen can handle.

It is useful to establish a convention linking one type of attention-getter with a particular subconcept. (e.g., the PL/I demonstration: blinking datum <=> new value; cursor <=> current instruction.) But even the best pointers cannot make up for a poorly structured presentation.

S.5 Change

Change, when it occurs, must not be too sudden. One needs to see the change in the process of happening. A motion involves a "from" and a "to" state as well as the movement itself: the context for the motion, if you will. An effective technique is to find a logical stopping place, in which the change is apparent and at the same time both the old and new states are visible.

In the PL/I demonstration, all the variables are displayed in the upper-right corner, with their current values shown next to the name. When an assignment is done, the old value is moved off to the right and the new one displayed, blinking for attention. At this point, the program pauses. The state of the screen allows full examination of the change. In the common situation in which the new value has been computed using the old one (in computing or summation), the display of the old value is essential for review of the action. When the old value is irrelevant, showing it allows the student to verify that irrelevance. (I find old values particularly disconcerting on a GET LIST input statement; but I feel it's important for the student to learn exactly how and when the values change.)

When the program continues from its pause, its very first action is to continue the blinking and remove the old value from the screen. I always tell my students that the (temporary) old value is just part of the demo, that in fact the old value is lost as soon as the assignment is made. This does not seem to cause trouble; the blinking new value seems to say, "temporarily frozen in mid-execution."

6. CONCLUDING REMARKS

The work reported above represents the first year's efforts from a continuing project. The PL/I demonstration system will be extended to the point where it can be called upon in as many as half the lectures in our introductory programming course. Many of the other projects should be improved, and new ones added to demonstrate additional dynamic concepts. Operating systems and numerical analysis contain a wealth of possible subjects, to name but two areas; I do not foresee a lack of candidates for projects.

All these projects are intended as lecture aids. They would also be most helpful in the consulting room, for personal counseling. Monetary factors (equipment) mediate against use by individual students; but even more important, I wish to avoid the quandary of creating a stand-alone instructional aid. The computer teacher tends to be seen first as a clever program, next as an operational problem, and only then as a teacher. A lecture aid, on the other hand, becomes an integral part of the user's presentation. It will not be used unless it conforms to his own teaching standards of clarity and conceptual soundness.

REFERENCES


APPENDIX 1: Student projects done in course CS 570 (Computers in Education), Spring, 1975.

3. Two-dimension array access: shows address calculation equation in operation, for absolute addresses. (Groppen).
4. Interrupt: shows interrupt-caused fetch and store of PSM, and restoration, using stylized machine code. (O'Reilly).
5. Lists and pointer-based variables in PL/I: shows allocation and data/pointer fields of based variables; interpretive system accepts program on-line. (Neuer).
6. Singly-linked lists: shows list traversal, node deletion and insertion, free storage, garbage collection, data access. (Meth).
7. Symbol table / Assembler: shows scan of input, operation of symbol table (lookup and insertion). (Kashinsky).
8. Stack: general stack operation using explicit push/pop commands from keyboard; also arithmetic expression parse into Polish using priority algorithm. (Mouril).
11. [Subsequent work] LISP: a) Shows list structure corresponding to an arbitrary input S-expression; b) Follows through execution of a LISP functional form, showing the precise progress of EVAL through the structure, including the recursive process of evaluating arguments which are themselves expressions. (Meth).

COUNT = 1
DATUM = 13 4
SPECSIZE = 15
SUM = 4.00

FIGURE 1. The PL/I Demonstration System in Action

This particular program reads in numbers and sums them until a specified total is reached. It was used to show yet another variety of iterative operation to the introductory PL/I course.

The formatting of the display is mostly automatic, except within statements, where the input spacing is kept. The variables, in the upper right, appear when their declaration is "executed"; SUM, which is not declared, was posted when first referenced (by the statement: SUM = 0); it appears as a FLOAT quantity in accordance with PL/I initial letter rules.

The arrow ("---") shows the statement which has just been executed; the asterisk on the next line indicates which statement will be next. Input and output appear at the bottom of the screen; one can see the latest value, 15, typed in response to the current GET LIST statement. The new
The value of 13 has been assigned to the variable DATUM, and is displayed in the upper right. Because the assignment has just occurred, the 13 is blinking; the former value of the variable, 4, has been moved aside. The program has paused, waiting for a carriage return. When it continues, the old value of DATUM will be removed from the screen and the current one will cease blinking.

FIGURE 2A: The LISP Demonstration System:
Display of an S-Expression in Structural Form.

LHEAD ( COND ( ( ZEROP N ) 21 ) ( T ( TIMES 3 ( DIFFERENCE N 6 ) ) ) )

FIGURE 2B: The LISP Demonstration System:
Execution of a LISP Functional Form

The program has just completed evaluation of the sub-expression (TIMES 3 5), and is in the process of replacing the whole sub-expression structure by the resulting value, 15.

The "•" marks represent list cells -- they appear fairly solid on the actual display unit. The colon next to PLUS and the heavy arrow ("---") indicate the progress of the evaluation at the higher level. (Project by Harry Meth, Summer 1975).
FIGURE 3  Demonstration of an Interrupt

An interrupt has happened at the fifth line of the main program; the word "INTERRUPT" is blinking. In response to the interruption, the current PSW is being saved in the reserved area "Saved PSW". The status ("4800") has already been saved. Preparatory to saving the location counter ("1008"), the demonstration program is "growing" a line from the "current PSW" towards the "Saved PSW". (Project by Patrick O'Reilly for CS570, Spring 1975.)

FIGURE 4  The FACET Machine in Operation.

The machine has just finished executing the "Load 10" instruction from location 2. The arrow at the accumulator is blinking for attention. (Project by Natalie Abrams for CS570, Spring 1975.)