INTERACTIVE VISUAL SIMULATORS FOR BEGINNING PROGRAMMING STUDENTS

Stuart C. Shapiro
Douglas P. Witmer

Computer Science Department
Indiana University
Bloomington, Indiana

1. Introduction

This paper discusses two programs that have been written to be aids to introductory programming students. They both embody the belief that Computer Assisted Instruction can be a worthwhile aid to students when properly used and that one of the best uses is to present visually to the student a process that he has some control over and which he would not otherwise be able to observe. Section 2 of this paper discusses HYCOMP1, an interactive visual computer simulator. Section 3 discusses IVF, the Interactive Visual FORTRAN interpreter. They were both written in SNOBOL4[1] and run under the KRONOS Time Sharing System on a CDC 6600 using an Applied Digital Data Systems, Inc. ADDS Consul 880 terminal, which is an ASCII terminal with a CRT display and an addressable cursor.

2. An Interactive Visual Computer Simulator[2]

2.1 Introduction

It is often felt desirable to include in an introductory computing course a small section of material on machine language, even when the course will mostly use a language such as BASIC, FORTRAN or ALGOL. The purpose of the section on machine language is to give the students an understanding of what a programmable, digital, sequential computer is, what its basic operations are, and how a program, made up of very basic operations can effect a fairly complicated calculation. To accomplish this purpose, it is not necessary to use a real computer as an example. Indeed, the complexities of a modern computer would obscure the points to be made. Therefore, various "hypothetical" computers have been invented that consist of a basic set of operations and registers. Students generally write programs for these hypothetical computers and find out if their programs are correct either from a human instructor or by executing them using a simulator in a batch environment. In either case, it is difficult for them to understand the sequential operation of their programs or the effect of any bugs that might have been in them.

In order to provide a better way for introductory computing students to gain this basic understanding, we have written an interactive simulator of one hypothetical computer. This simulator displays all the storage and active registers of the hypothetical computer on the screen of a CRT terminal, allows the student to load and patch programs, load data, and execute the program. The simulator traces the program by changing the information on the screen at a slow enough rate that the student can watch how his program works. He can then make any modifications he feels are necessary and see the effect of those changes.

2.2 HYCOMP1

The simulator, HYCOMP1 was written to simulate the HYCOMP computer of Terry Walker's introductory text.[3] Walker's HYCOMP has 1000 words of memory, each holding a sign plus five decimal digits. Its instruction code ignores the sign, uses the two high order digits for an operation code and the remaining three digits for an address. In order to display all of memory on the CRT, we implemented only 100 words for HYCOMP1, and use the two low order digits for the address, ignoring the middle digit. HYCOMP and

---

[1] Griswold, R.E., Poage, J.F., Polonsky, I.P., The SNOBOL4 Programming Language, 2nd edition, Prentice-Hall, Englewood Cliffs, N.J., 1971.

[2] HYCOMP1 was programmed by David A. Grace.

[3] Walker, Terry M., Introduction to Computer Science: An Interdisciplinary Approach, Allyn and Bacon, Inc., Boston, 1972.

HYCOMP1 have the following active registers: a sign plus 5 digit arithmetic register (REG); a 5 digit control unit (CU); a program instruction counter (IC) which is three digits in HYCOMP and two digits on HYCOMP1. They also have an overflow (OF) switch, underflow (UF) switch, and an end-of-file switch (EOF). Floating point numbers are stored using the two high order digits as a biased exponent. There are forty nine alphanumeric characters coded two digits per character and stored two characters per word in the low order four digits. The 24 instructions include numeric and alphanumeric I/O, integer and floating point arithmetic and test and branch instructions. HYCOMP does not have index registers, but in the future, they may be added to HYCOMP1 using the middle digit of instructions for the index register field.

2.3 Using HYCOMP1[4]

When the student executes HYCOMP1, the screen is cleared and written on as shown in fig. 1. He may then load his program (only machine language is accepted by HYCOMP1) and his data and run the program. He may then patch or change his program and continue trying and changing it until he is satisfied or tired. The HYCOMP1 simulator gives the student the same options he would have if he were sitting at a HYCOMP1 console, but in addition, he can observe the execution of his program.

There are two commands with which the student can execute his HYCOMP1 program - RUN and CYCLE. With RUN, the student specifies the starting address of his program, and it is executed until it halts or produces an execution error. If the student uses CYCLE, one machine cycle is executed so that he can study the results of each instruction at his own speed.

A machine cycle consists of the following steps:

1a. The CRT cursor underlines the contents of the word whose address is in the IC.
b. That instruction is copied into the CU and displayed there on the screen.
2. IC is incremented by 1, the new value being displayed properly on the screen.
3. The instruction in the CU is executed. This may cause various things to happen on the screen, for example:
a. A test instruction causes the displayed contents of REG to blink for a short time.
b. A branch instruction changes the IC.
c. Whenever data is read, either from REG, a memory word or on the Input line, the cursor underlines the data.
d. Data may be written into REG, a memory word or the Output line.
e. A floating point instruction may set

```
00:      01:      02:      03:      04:      05:      06:
07:      08:      09:      10:      11:      12:      13:
14:      15:      16:      17:      18:      19:      20:
21:      22:      23:      24:      25:      26:      27:
28:      29:      30:      31:      32:      33:      34:
35:      36:      37:      38:      39:      40:      41:
42:      43:      44:      45:      46:      47:      48:
49:      50:      51:      52:      53:      54:      55:
56:      57:      58:      59:      60:      61:      62:
63:      64:      65:      66:      67:      68:      69:
70:      71:      72:      73:      74:      75:      76:
77:      78:      79:      80:      81:      82:      83:
84:      85:      86:      87:      88:      89:      90:
91:      92:      93:      94:      95:      96:      97:
98:      99:       OF=    UF=    IC=      CU=          REG=
                  EOF=
TO SEE A LIST OF AVAILABLE COMMANDS, TYPE HELP
TO PROCEED AFTER AN ERROR MESSAGE APPEARS, HIT  NEW LINE
TO LEAVE A MODE, JUST TYPE NEW COMMAND.  ? MEANS READY
INPUT APPEARS ON LINE 22, OUTPUT ON LINE 23
?_
```

Fig. 1. CRT screen after initialization

[4] A user's guide is available as: Shapiro, Stuart C. and Grace, David A., "A Guide to the Use of HYCOMP1", Technical Report No. 8, Computer Science Dept. Indiana Univ., Bloomington, In., Dec. 10, 1973.

UF or OF.

f. An input instruction may set EOF.

If an execution error occurs, an appropriate message is written on the screen. When the student is ready, he can clear this message and enter additional commands.

The valid commands are:

±nnnnn - The signed five digit number is put into the word whose address is the current value of IC and IC is incremented by 1. If the sign is omitted, + is assumed.

LOAD nn - IC is set to nn, OF, UF, and EOF are initialized to 0, and the input line is positioned at the first "card" of data. If nn is omitted, 00 is assumed.

RUN nn - IC is set to nn and the program is run to termination or until an execution error occurs. If nn is omitted, the current value of IC is retained.

IC XX - XX is a two digit number. IC is set to XX.

UF X - X is 0 or 1. UF is set to X.

OF X - X is 0 or 1. OF is set to X.

EOF X - X is 0 or 1. EOF is set to X.

REG n - n is an optionally signed integer with no more than five digits. REG is set to n.

CY - One machine cycle is executed.

QUIT - The HYCOMP1 simulator is terminated.

DATA - Allows data to be entered, one "card" at a time. A card with # in column 1 signals the end of data entry and serves as the EOF flag.

DATA ADD - Allows data to be added to the end of the existing data "deck". The existing EOF card is automatically removed.

HELP - A list and brief description of the commands is displayed.

2.4 Summary

HYCOMP1 has not yet had extensive classroom use. We did have several students from an introductory programming class use it during the latter stages of its development. Earlier in the semester, these students had studied HYCOMP and had written several HYCOMP programs which were graded by a human grader. Their reaction to HYCOMP1 was extremely favorable and they reported that it would have been very helpful in their study of machine language. One student finally understood that there is no inherent difference between instructions and data. This kind of insight, if not gained intellectually, could hardly be gained by any means other than an interactive, visual simulation.

3. The Interactive Visual FORTRAN Interpreter

3.1 Introduction

Learning to program in a higher level language such as FORTRAN can be a formidable task for the beginning student. He must not only memorize the key words and statement syntax associated with the language, but must learn to assemble meaningful programs from these building blocks. In this regard, it is essential for the student to be able to visualize the step-by-step operation of his programs. The Interactive Visual FORTRAN Interpreter (IVF) was designed as an aid to such a visualization.

3.2 Basic Description

The IVF accepts a FORTRAN program and graphically simulates its execution on an interactive time-sharing terminal with a cathode ray tube (CRT) screen. During the simulation the executable statements are displayed on the screen. An arrow is caused to point at each statement in the order in which execution would normally take place. Current values of variables are displayed in the unused space to the right of the program, and are updated with each move of the arrow. Thus the immediate state of the program is visually apparent at all times in terms of current values of variables and flow of control.

Figure 2 shows a short program as it would appear during simulation on the IVF. The arrow indicates that execution of the statement "X = (I ** 2) / 2" is being simulated. The current values of "I" and "X" are displayed on the right side of the screen. In this example the arrow will next move to the "IF" statement. Simulation of the "IF" statement succeeds and the arrow moves to the statement whose label is "4". Now the value of "I" is incremented and the new value is displayed on the right. The simulation proceeds in this manner until execution of the "IF" statement is simulated with the current value of "I" equal to 10. At this time the arrow moves to the "END" statement and the simulation is terminated.

Upon termination of the execution phase, the IVF displays an appropriate message and the user is given the option to exit the IVF or attempt another simulation. The FORTRAN program just simulated is not lost in either case. The user may have it simulated again (as discussed later) or dispose of it in some other fashion.

The IVF assumes a naive user, and is therefore self-explanatory. Instructions may be received at the beginning of each session by responding appropriately to the program's inquiries. In addition, all portions of the program requiring a user response will recognize errors and prompt as necessary.

Although the IVF is primarily an aid to the visualization of FORTRAN program execution, it is also an interactive FORTRAN statement accepter with a number of error checking facilities. The IVF will accept FORTRAN lines and continuation lines until a complete statement has been entered. It then checks for syntax errors,

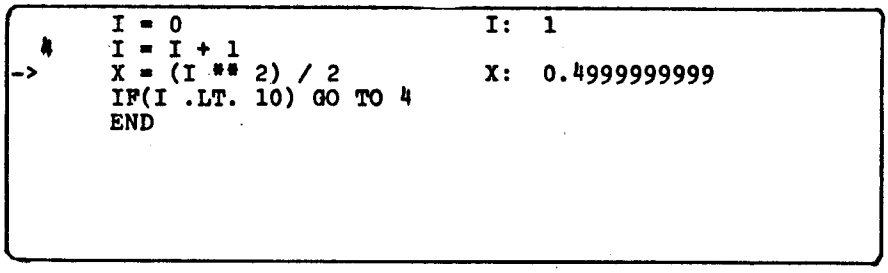

Figure 2

errors in "DO" loop nesting, and multiply defined statement numbers. If the IVF detects an error, the statement is rejected and an appropriate error message is displayed. The user is thus made aware of his mistake immediately and is given another opportunity to enter a correct statement.

3.3 Additional Features

A user may optionally bracket a portion of his FORTRAN program by placing a "+" in the continuation field of a noncontinuation line to turn the "STEPPED MODE" on. The "STEPPED MODE" is turned off in an analogous fashion with a "-". Within the bracketed section the user must press the "RETURN" key after the execution of each FORTRAN line has been simulated. This permits a slowing of the simulation over a portion of the user's program for more careful study.

The IVF saves a copy of the FORTRAN program simulated, and makes it available on a file reserved for that purpose. The user may optionally submit complete programs to the IVF rather than enter them one line at a time. Error checking is performed on each statement as with on-line entry. The complete program may have originated from any source, including a session with the IVF.

3.4 Implementation

The IVF is written in the SNOBOL4 programming language. Each FORTRAN statement accepted by the IVF is translated into a sequence of SNOBOL statements. Steps are taken to adjust operator precedence and make type conversions as necessary. This is bracketed by statements which move the arrow and update displayed information during the simulation. This block of SNOBOL statements is concatenated into a string of such blocks for later compilation, utilizing the CODE function.

Simulation of a FORTRAN program is accomplished by first displaying its executable statements on the CRT screen, and then executing the compiled SNOBOL statements discussed above. During simulation the IVF displays output from the FORTRAN program on the bottom line of the screen, and accepts input on the line above. The position of the executable FORTRAN statements on the CRT screen is never changed. The CRT's cursor (which underlines the next printable position) is moved about as necessary to prevent erasure of valuable information and to facilitate input and output.

3.5 Summary

Visualization of the execution of his programs is of utmost importance to the beginning programmer. It is hoped that the IVF will serve as an effective aid to such visualization. It is further hoped that the IVF will be found flexible in its features and yet simple enough for use by a beginning programmer.

IVF can readily be adapted to run on a number of computing systems which support time-sharing terminals and provide interactive execution. The primary restrictions are the need for a CRT display at the time-sharing terminal and a sufficiently large character set to support control of the display.

The IVF is written in a straightforward manner with numerous comments. An experienced programmer should, therefore be able to expand the scope of the IVF to suit the needs of his local users. He might even cause the IVF to simulate some language other than FORTRAN.