Behavior Research Methods & Instrumentation
1978, Vol. 10 (6), 764-772

PLE: A high-level multiprogramming language for psychology

JOHN C. PALMER
University of Michigan, Ann Arbor, Michigan 48104

COLIN M. MACLEOD
Scarborough College, University of Toronto, West Hill, Ontario, Canada

and

GEOFFREY R. LOFTUS
University of Washington, Seattle, Washington 98195

This paper describes a software system called PLE that is designed to turn a Data General Corporation computer system into a sophisticated infinite-channel tachistoscope. We describe hardware and software characteristics of the PLE system and evaluate its performance in comparison to a typical tachistoscope. Additionally, we describe two example experiments that have been implemented in the PLE system.

Several characteristics of a typical tachistoscope might be considered when comparing an actual tachistoscope with a computer system as tachistoscope. First, a tachistoscope affords fairly precise control over stimulus timing. Similarly, a tachistoscope generally affords fairly precise control over stimulus luminance. Third, a tachistoscope allows the use of any visual stimulus that can be printed on a piece of paper. Fourth, a tachistoscope is (in practice) limited in the number of channels that can be independently presented. Fifth, a tachistoscope is limited with respect to the flexibility of stimulus presentation order and response collection. And sixth, a tachistoscope is generally limited in the sense that it can be used to run only a single subject at a time.

In this paper, we describe a programming language (PLE, an acronym for programming language for experimentation) developed at the University of Washington (cf. Burkhardt, 1976, Note 1, for more complete descriptions). PLE is a high-level, FORTRAN-like, easy-to-learn-and-use language whose purpose is to turn a computer system into a highly sophisticated tachistoscope. Our description will compare the PLE system with a tachistoscope, focusing on the various characteristics of tachistoscopes enumerated above. The paper has four major sections. We first describe the hardware environment within which PLE is currently running. Second, we describe some of the principal characteristics of the PLE software itself. Third, we provide two examples of standard experimental paradigms as they are implemented in PLE. And, finally, we remark on the performance characteristics of PLE.

We are grateful to Zelda Zabinsky for her helpful criticisms during the preparation of this report. Principal credit for the development of the PLE language goes to Ken Burkhardt, who is now at the Department of Electrical Engineering, Rutgers University. The development of PLE was supported by the National Institute of Mental Health, Grant MH 21795, Earl Hunt, principal investigator, and the National Institute of Education, Grant NIE-6-74-0104, Clifford Lunneborg, principal investigator. The writing of this paper was supported by National Science Foundation Grant BNS 76-23367, Geoffrey Loftus, principal investigator. Reprint requests may be sent to Geoffrey R. Loftus, Department of Psychology, NI-25, University of Washington, Seattle, Washington 98195.

Visual Displays

We will concentrate here primarily on an arrangement that uses a computer-controlled cathode-ray tube (CRT) as a stimulus display device. We note briefly, however, that the computer can also be used to control a slide projector equipped with a tachistoscopic shutter. As currently implemented on the University of Washington system, there is one on-line projector/shutter in each experimental booth. Such a configuration is useful in a situation where complex visual stimuli (e.g., color photographs) are used. Timing is quite good: Rise and fall times of the shutters are about 1 msec. However, this configuration loses power with respect to luminance control, interval timing, number of potential fields, and flexibility of stimulus presentation.

HARDWARE REQUIREMENTS

PLE's use is restricted to Data General Corporation computers because of large amounts of machine-dependent code. However, the PLE language is currently implemented on two different systems and could be useful in a variety of peripheral environments. The common elements of the systems are 24K memory NOVA minicomputers supported by Ball Computer disk systems and software. The two systems both use Tektronix 604 scopes with the fast-decay P-11 phosphor, but differ in their scope controller hardware and software. The PLE language was initially designed

on a system with a powerful IKON scope controller (Burkhardt, Palmer, & Waddington, 1976) and later was extended to operate on a second system's more primitive, custom-built controller. The IKON system generates and maintains character displays completely with hardware; the second system requires software to convert characters into coordinates and to maintain the display. Because of PLE's modular design, the extension to the more primitive system was reasonably simple and only occupied a fraction of the original implementation time. Thus, while PLE is currently implemented on quite specific equipment, it can be modified to fit other Data General systems.

SOFTWARE

The usefulness of the computer as a tachistoscope is determined primarily by the flexibility and usability of its software. To optimize flexibility and usability, PLE includes common FORTRAN-like commands as well as special-purpose commands needed in tachistoscopic experiments. The latter commands include formatting for fast-phosphor CRTs, millisecond timing functions, and multiprogramming capabilities to asynchronously control eight stations.

Basic Concepts

Data types. Several data types exist in PLE. First, explicit data types are defined along three binary dimensions: Variables and constants are either integers or strings, variables are either scalars or vectors, and variables are either global or local. Briefly, global variables refer to all experimental stations, whereas local variables refer to individual stations, a distinction that will be described in more detail below. By default, variables are global scalar integers, but may be declared any combination of the possible data types.

Display graphics are represented as an implicit data type. Line endpoints can be used to generate strings of graphic characters, which in turn can be output in strings to display the line graphics. This representation allows an interchangeable treatment of alphanumeric and graphic strings. In practice, this means the same programs can display sentences or line drawings as stimuli. For a further treatment of PLE graphics, see Burkhardt et al. (1976).

Operations. A number of operations are possible on both integer and string data types. Integer operations include the usual arithmetic functions: addition (+), subtraction (–), multiplication (*), and division (/). String operations include string assignment (SASSIGN), string addition (CATONATE), and string subtraction (SUBSTR). In addition, there are a few special-purpose operations, including a random-number generator (RAND), a vector permutation command (PERM), logical "and" (LAND), logical "or" (LOR), and a routine to define endpoint graphics (GRAPH).

Branching statements. The branching statements are similar to those used in FORTRAN and ALGOL. A loop control command (DO), directional statements (GOTO), and comparison statements (CMP—compare two integers) are analogous to the FORTRAN DO loop, GO TO, and IF statements, respectively. In addition, a convenient ALGOL-like IF-THEN-ELSE-DONE branching command, coupled with subroutine calls (CALL), permits use of a highly efficient block structure (see Visual Search Example, below).

Asynchronous control. The most novel feature of PLE is its multiprogramming capabilities. PLE can execute a program in each of eight stations with little additional programming effort and minimal decrease in timing accuracy. These eight stations are controlled through a single program that specifies the procedures and data structures that are to be shared and those that are to be independent for each of the stations. Programming simplicity results from restricting PLE's multiprogramming to one of two modes, global or local. Global mode restricts the program to a single execution of the procedures and data common to all stations, while local mode allows independent execution for each of the independent stations. Execution mode is easily controlled by the user. Programs initially enter global mode, but may be placed into local mode by the FORK command and later returned to global by the JOIN command. In a similar fashion, every variable is classified as either a global variable or a local variable. Local variables and station-specific I/O devices (CRTs and response keys) can be accessed only in local mode. Global variables and common I/O devices (TTY and disk) can be accessed from either mode.

Formatted I/O

A PLE I/O statement consists of a command word, a format literal or a label referencing a format statement, and an associated variable list. The command word selects a particular I/O device, such as the CRT output (WRITSCOPE) or the Teletype keyboard input (READTTY). Individual I/O operations are specified with format characters accompanied by idiosyncratic arguments and associated variables. A combination of format characters followed by arguments enclosed in carets constitutes a format literal, or format statement. Examples of several format characters, together with their functions, arguments, and associated variables, are presented in Table 1.

As an illustration, to output the string constant "HELLO" to a scope, the following command is used:

```
WRITSCOPE "Z⟨0⟩ A⟨5⟩" "HELLO"
```

In this command, the character size has been set to the large size (0 in the Z format) and the variable has been specified as alphanumeric (the A format). The digit 5 in the argument following the A format sets the field length of the associated variable to five characters; a

Table 1
Partial List of Format Commands

Formatting Function	Character	Argument	Associated Variable
Expect an alphanumeric variable	A	Field length	String variable
Close a disk file	C	File number	None
Expect a double-precision integer	D	Field length	Integer
Reference a disk file	F	File number	None
Open (get) a disk file	G	File number	File name string
Expect a single-precision integer	I	Field length	Integer
Clear CRT screen	K	None	None
Output a line (CR,LF)	L	Number of lines	None
Purge all scope storage	P	None	None
Rewind a disk file	R	File number	None
Output an alphanumeric constant	W	"String"	None
Output spaces	X	Number of spaces	None
Set character size	Z	Size	None

field length of zero would allow free formatting. Free formatting is convenient in scope displays where the size of stimuli may vary widely; however, fixed formats are required for disk and TTY output.

More complex displays may be constructed with equal ease. As an instance, a number-word paired associate can be output to the center of the screen by defining the following formats: the character size (Z), the number of lines (L), the number of spaces (X), an integer variable (I), and string variable (A). Consider the command:

```
WRITSCOPE "Z⟨0⟩ L⟨7⟩ X⟨4⟩ I⟨2⟩ X⟨2⟩ A⟨0⟩" NUMBER WORD
```

where the variables NUMBER and WORD might be "89" and "SHIRT." This will produce an image on the CRT that is seven lines down from the top, four spaces to the right, the number 89, two spaces, and the string "SHIRT." Additional formats are available for the special functions of scopes, disks, and Teletypes. For example, to erase a scope, the clear screen (K) and purge scope memory (P) formats must be used, as:

```
WRITSCOPE "K P"
```

The formats already presented are sufficient for most experiments.

Timing Statements

There are a number of ways to time events using the PLE language. A millisecond clock is available to the user in the reserved double-precision variable, TIMEO, but it is usually more convenient to use one of four special timing commands. The timing commands control the program by delaying execution for a specified number of milliseconds, delaying execution until a particular keypress is made and recording that response and reaction time, recording a response and time without delaying program execution, or delaying execution until either a response or a specified amount of time has passed.

In local mode, the basic timing command (WAIT) delays program execution at a particular station for the number of milliseconds specified by its single argument. For example, this command may be used to control the duration of a CRT display:

```
WRITSCOPE     "A⟨0⟩"     "READY"
WAIT 200
WRITSCOPE     "K P"
```

This sequence displays "READY," waits 200 msec, and then clears the screen.

Three alternative commands, WAITK, RECDK, and WAITR, record keypresses and reaction times. These commands each require three arguments: the desired key(s), the key actually pressed, and the reaction time. The desired key variable must be defined in the program, while the other two will be returned to the program when the subject presses a key. Keypresses are identified by a convention whereby each of the eight least significant bits represents a separate key response (e.g., 1 = right key, $200_8 = 10\ 000\ 000_2$ = left key, and $377_8 = 11\ 111\ 111_2$ = all eight keys). This code is used to represent both the desired key and the response variables. A reaction time is represented as a double-precision integer that must be stored in two successive elements of an integer vector. For the reaction time variable, RT(0), the most significant half of the reaction time is stored in RT(0) and the least significant half is stored in RT(1). This requires that RT be dimensioned to two.

The wait-for-keypress command (WAITK) delays further execution of the program until a desired key is pressed. The WAITK command has the three basic arguments already described: a desired key variable, a reaction time variable, and a response key variable. In the following example, the three statements display a message to the subject until he or she presses any key, and the scope is then cleared.

```
WRITSCOPE   "A ⟨0⟩"   "PRESS ANY KEY WHEN READY"
WAITK 377   RT(0)     RESP
WRITSCOPE   "K P"
```

The record-keypress command (RECDK) does not delay program execution while recording the desired keypress and reaction time. Instead, it permits program execution to continue uninterrupted. To detect when a

Table 2

```
RECDK 200 RT (0) RESP
DO XX. FOR I = 1, 20
WRITSCOPE "A<0)" STIM(I)          ; display the Ith stimulus.
WAIT 100                          ; wait for 100 msec.
WRITSCOPE "K P"                   ; clear screen.
IF (RESP. NE. 0)                  ; check if a response has been made.
   THEN GOTO YY.                  ; yes, go on and process it.
   ELSE                           ; no continue with stimulus sequence.
   DONE
XX: CONTINUE                      ; continue with do loop.
YY:
```

response is finally made, the program must check if the response variable has been changed from its initially zeroed state. RECDK's arguments are identical to those of WAITK. To illustrate RECDK's potential, the example shown in Table 2 displays a sequence of stimuli until the leftmost key is pressed.

The fourth timing command (WAITR) waits until a response is made or a time criterion is past and branches as a result of which of these two events occurred. WAITR, like WAITK and RECDK, also records the response and reaction time. It has six arguments; the desired keypresses, a time criterion, a reaction time, a response, a keypress-exit label, and a time-up-exit label. This command may be used to limit the time that a program will wait for a keypress. Further examples of these timing commands may be found in the sample experiments below.

TWO EXAMPLES

To illustrate the use of PLE in a tachistoscope setting, we present two sample programs. The first is the Sternberg (1966) memory scanning task, the standard example in this collection of papers. Following this is a more complex program, the Sperling (Sperling, Budiansky, Spivak, & Johnson, 1971) high-speed search task. These examples are provided to develop some of the basic PLE properties introduced in prior sections of this paper.

The Memory Scanning Example

The Sternberg (1966) paradigm is used for examining time to scan brief lists in short-term memory. In the "fixed-set" version presented here, the subject memorizes a short list of from one to five digits read by the experimenter. Following memorization, a single probe digit is presented on the CRT. As soon as possible, the subject presses a key to indicate whether that digit is or is not in the memory set.

By looking at mean reaction time to probes for memory set sizes of from one to five digits, the time to scan each digit in the set can be estimated. For the average college student, each extra digit in the set adds between 30 and 40 msec of scanning time. This is true regardless of whether the probe digit is or is not in the set, suggesting that subjects always search the memory set exhaustively.

The PLE program implementing this simple task is shown in Figure 1. Basically, the program selects and displays a probe digit to each subject, after a warning that the trial is beginning. When a subject responds, the data are output to the disk in the form of the probe

Figure 1

```
;------------------------  DECLARATION STATEMENTS  ------------------------

        DIMENSION BUF(10),RT(2)          ;BUF IS THE BUFFER AREA TO HOLD THE
                                         ;NAME OF THE STRING VARIABLE FILEN.
                                         ;RT NEEDS DIMENSIONING (IN OCTAL) TO
                                         ;SIMULATE A DOUBLE PRECISION INTEGER.
        STRING FILEN                     ;SET UP FILEN AS A STRING

        LOCAL TRL,SEED,S,RT,RESP         ;FOR EACH OF THESE VARIABLES, SUBJECTS
                                         ;MUST HAVE INDEPENDENT VALUES.

;------------------------  EXECUTABLE STATEMENTS  ------------------------
;INITIALIZE
        READTTY "W'FILENAME'A<0>" FILEN BUF(0)  ;ACCEPT FILENAME FROM TTY
        WRITDISK "G<1>" FILEN            ;OPEN FILE #1 ON DISK FOR DATA STORAGE.

        READTTY "W'SEED'I<0>" GSEED      ;ACCEPT FROM TTY A GLOBAL SEED TO
                                         ;INITIALIZE RANDOM NUMBER GENERATOR
        FORK                             ;ENTER LOCAL MODE FOR SUBJECT SPECIFIC
                                         ;COMMANDS.
        SEED=GSEED                       ;SEED MUST HAVE LOCAL VALUES OTHERWISE
                                         ;RANDOM SEQUENCE COULD NOT BE REPEATED.
;EXECUTE 100 TRIALS OF THE EXPERIMENT
        DO ETRL   FOR TRL=1,100          ;ETRL  IS A LABEL (DESIGNATED BY  )
```

```
                                        ;100  IS DECIMAL (ALSO DESIGNATED BY .
                                        .NUMERIC CONSTANTS ARE OCTAL BY DEFAULT
        S=RAND(SEED,9 )                 .SET STIMULUS TO A RANDOM NUMBER 0 TO 9

        WAIT 500                        .WAIT 500 MS  BETWEEN TRIALS

        WRITSCOPE "Z<0>L<7>X<7>A<0>" "+" .DISPLAY A LARGE "+" AS A WARNING
                                        .AT THE CENTER OF THE SCREEN

        WAIT 500.                       ;WARNING INTERVAL

        WRITSCOPE "KP"                  ;ERASE WARNING STIMULUS.

        WRITSCOPE "Z<0>L<7>X<7>I<1>" S  ;DISPLAY PROBE DIGIT AT SCREEN CENTER

        WAITK 201 RT(0) RESP            ;WAIT FOR RESPONSE FROM LEFT OR RIGHT
                                        .KEY ONLY (SET BY 201, AN OCTAL MASK).
        WRITSCOPE "KP"                  ;ERASE SCREEN

ETPL:   WRITDISK F  INSUB S RT(1) RESP  .OUTPUT PROBE DIGIT, RT, AND KEY NUMBER
                                        ;TO DISK   INSUB CONTAINS SUBJECT
                                        ;IDENTIFIER   FORMAT IS AT F: BELOW
;CONCLUDE PROGRAM
        JOIN                            .WAIT FOR ALL SUBJECTS TO FINISH
                                        .EXPERIMENT
        WRITDISK "C<1>"                 .CLOSE DISK FILE

        EXIT                            .RETURN TO SYSTEM MONITOR

;------------------------  FORMAT STATEMENTS  ------------------------

F       FORMAT(F<1>I<7>I<7>D<7>I<7>L<1>)  ;DISK OUTPUT FORMAT

        END
```

Figure 1. A sample experiment illustrating a fixed-set version of the memory scanning task.

presented, the key pressed, and the reaction time for that trial. Each line of output is identified with a station number.

This simple experiment performs the three functions common to most tachistoscopic experiments: stimulus selection, display generation, and response collection. The stimulus for each trial is chosen randomly using a pseudorandom-number generator (RAND). The display sequence consists of a fixation field, warning interval, erase command, a number stimulus field, response interval, and final erase command. Each of these steps is accomplished by a single statement. The responses are collected by the command WAITK, which records the response and reaction time, after delaying program execution until the response is made. The response and reaction time are then written to the disk along with the stimulus presented and a station identifier. To replicate this experiment on a manual tachistoscope would require three fields and three timers, in addition to the response collection equipment.

The Visual Search Example

The second sample program is considerably more complex. It represents the high-speed visual search task introduced by Sperling et al. (1971) and recently modified by Schneider and Shiffrin (1977). This task is used to determine how quickly subjects can search for particular symbols. As is shown in Figure 2, a target stimulus appears before the search set is presented to the subject, rather like reversing the Sternberg (1966) paradigm. The search set consists of a series of visually presented arrays (or frames), one of which may contain the target stimulus. The critical manipulation in this example program is the time interval between successive frames on a given trial; this interval is varied across blocks of trials by inputting intervals from the Teletype. Longer interframe times permit more of the stimuli on a given frame to be searched. Unlike the Sternberg task, both the speed of the response and its accuracy are of interest in this task. Accuracy becomes a crucial dependent variable when interframe times are short and subjects cannot search all of the elements of the display.

This experiment performs the same kinds of functions as the memory scanning experiment, but in a considerably elaborated manner. Three routines are involved in stimulus specification and selection. The first is the initialization (INIT) subroutine: Among its several functions, this routine reads a stimulus file and calculates the stimulus display positions. Another subroutine (BALANCE) sets up a vector specifying a counterbalanced sequence of target and distractor trials. The first half of the third subroutine (TRIAL) determines from the counterbalancing vector which trials will include the target. TRIAL then randomizes target

Figure 2

```
;PROGRAM TO PRESENT VARIED-MAPPING, MULTIPLE-FRAME VISUAL SEARCH TASK
;------------------------  DECLARATION STATEMENTS  ------------------------
DIMENSION COND(144),INDEX(144),BUF(100),PC(2),PT(2),SBUF(4704)
DIMENSION ISI(12),STIM(20),LOC(5),XPT(5),YPT(5)
STRING STIM,SNAME,DUMMY
LOCAL   RESP,TARG,PT,PC,BLK,TPL,NTPL,I,J,SEED,COND,INDEX,CRIT,REP,LOC
;-------------------------  EXECUTABLE STATEMENTS  -------------------------
;BASIC PARAMETERS
        DUR=12                          ;12 MS SUFFICIENT FOR 2 REFRESHES
        NDTPL=10                        ;NUMBER OF DATA TRIALS
        NBLK=4                          ;NUMBER OF BLOCKS
;MAIN PROCEDURE
        CALL INIT                       ;CALL INITIALIZATION SUBROUTINE
        FORK
        DO EBLK,  FOR BLK=0,(NBLK-1)    ;START BLOCK LOOP.
        NTPL=10                         ;PASS THIS ARGUMENT
        CALL BALANCE
        DO PTPL   FOR TPL=0,1           ;START PRACTICE LOOP.
PTPL:   CALL TRIAL
        WRITSCOPE "Z<200>L<2>A<0>" "END OF BLOCK"
        PC(0)=0                         ;RESET THESE SUMS TO ZERO
        PC(1)=0
        NTPL=NDTPL                      ;PASS ARGUMENT
        CALL BALANCE
        DO ETPL   FOR TPL=0,(NDTPL-1)   ;START EXPERIMENTAL TRIALS.
ETPL:   CALL TRIAL.
        WRITSCOPE "Z<200>L<2>A<0>" "END OF BLOCK"
;WRITE OUT DATA SUMMARY, INCLUDES SUBJECT ID, ISI, HITS, & FALSE ALARMS.
EBLK:   WRITTTY "I<7>I<7>I<7>I<7>L<1>" INSUB ISI(BLK) PC(0) PC(1)
        JOIN
;FINISH UP
        WRITTTY "A<0>L<1>" "PROGRAM COMPLETE"
        EXIT

;INIT SUBROUTINE   SETS UP STIMULI AND INPUTS PARAMETERS.
INIT:   READTTY "W'STIMULUS FILE'A<0>" SNAME BUF(10)    ;GET STIMULUS FILE.
        READDISK "G<0>I<7>X<1>I<7>X<1>I<7>X<1>" SNAME NSTIM VSIZE HSIZE
        DO INIT1.   FOR A=0,(NSTIM-1)   ;READ IN EACH STIMULUS
INIT1:  READDISK "F<0>A<0>" STIM(A) SBUF(A*23 ) ;39 CHAR MAX
        WRITTTY "A<0>" "ISI LIST"
        DO INIT2   FOR A=0,(NBLK-1)     ;READ IN EACH ISI VALUE
INIT2.  READTTY "I<0>" ISI(A)
        READTTY "W'SEED'I<0>" GSEED
        FORK
        SEED=GSEED                      ;CREATE LOCAL SEED
        DO INIT3   FOR I=0,(NSTIM-1)    ;DEFINE INDEX VECTOR FOR LATER USE.
INIT3:  INDEX(I)=I                      ;WILL BE USED TO SELECT STIMULI.
        JOIN
        A=0                             ;SET UP POSITION PARAMETERS.
        DO INIT4.   FOR X=0,1           ;DEFINE SCREEN LOCATIONS FOR
        DO INIT4.   FOR Y=0,1           ;EACH DISPLAY POSITION.
        A=A+1
        XPT(A)=7-VSIZE+X*(VSIZE+1)      ;VERTICAL SPACING
INIT4:  YPT(A)=7-HSIZE+Y*(HSIZE+1)      ;HORIZONTAL SPACING
        WRITTTY "A<0>L<1>" "   SUBJ    ISI    HITS    FAS"       ;TABLE HEAD.
        RETURN

        FORK                            ;LOCAL SUBROUTINES FOLLOW.
;BALANCE SUBROUTINE   PASS IT NTPL AND IT RETURNS PERMUTED CONDITION VECTOR.
BALANCE: DO BAL1.   FOR I=0,(NTPL-1)
BAL1:   COND(I)=I/(NTPL/2)              ;SET FIRST HALF TO 0, SECOND HALF TO 1.
        PERM COND(0) NTPL SEED          ;PERMUTE THIS BINARY LIST.
        RETURN

;TRIAL SUBROUTINE   PASS IT TPL,COND,INDEX,STIM,VSIZE,HSIZE,XPT,YPT,DUR,ISI,
;BLK,NSTIM,SEED AND IT EXECUTES A TRIAL AND STORES RESULT IN PC
TRIAL   WRITSCOPE "Z<200>L<14 >X<12 >A<0>" "READY"      ;GIVE WARNING.
        WAITF 377 PT'0, RESP            ;WHEN READY, SUBJECT PRESSES ANY KEY.
        WRITSCOPE "I P"
        PERM INDEX'0, NSTIM SEED        ;CHOOSE TARGET.  (INDEX(0))
        TARG=INDEX'0
        CRIT=RAND(SEED,6)+2             ;CHOOSE FRAME TO DISPLAY TARGET.
```

```
;DISPLAY TARGET IN CENTER OF SCREEN
;NOTE THAT FORMAT ARGUMENTS OF 377 INDICATE THAT THE ACTUAL ARGUMENT
;WILL APPEAR IN THE ASSOCIATED VARIABLE LIST
        WRITSCOPE "Z<0>L<377>X<377>A<0>" <7-VSIZE/2> <7-HSIZE/2> STIM<TARG>
        WAIT 1000                       ;LET SUBJECT STUDY TARGET FOR A SECOND
        WRITSCOPE "KP"
        WAIT 250
        RECDK 201 PT<0> RESP            ;START RESPONSE RECORDING ROUTINE
        DO TRL1.  FOR REP=1,10          ;NOW PRESENT 10 VISUAL ARRAYS
        IF <RESP  NE  0>                ;CHECK IF RESPONSE YET
          THEN  GOTO FEED               ;YES SKIP AHEAD
          ELSE                          ;NO
          DONE
;SET UP VISUAL ARRAYS   USE ALL DISTRACTORS, EXCEPT ON TARGET TRIALS.
        IF<REP  EQ. CRIT> .AND  <COND<TRL>  EQ. 0>> ;TEST IF TIME FOR TARGET.
          THEN  LOC(1)=INDEX<0>         ;YES. PLACE TARGET INDEX INTO LOC
          ELSE  LOC(1)=INDEX<1>         ;NO. INSERT FIRST DISTRACTOR INSTEAD
          DONE
        DO  TRL2   FOR I=2,4            ;CHOOSE REST OF DISTRACTORS.
TRL2:   LOC<I>=INDEX<I>
        PERM LOC(1) 4 SEED              ;PERMUTE POSITION OF STIMULI
        DO TRL3.  FOR I=1,4             ;LOAD EACH STIMULUS.
;LOAD STIMULI INTO SCOPE SOFTWARE STORAGE WITHOUT DISPLAYING THEM <U>.
;NOTE:  THE <LOC<I>+0> EXPRESSION IS NECESSARY BECAUSE PLE DOES
;NOT ALLOW VECTOR AS SUBSCRIPTS BUT DOES ALLOW EXPRESSIONS AS SUBSCRIPTS
TRL3.   WRITSCOPE "Z<0>L<377>X<377>A<0>U" XPT(I) YPT(I) STIM<LOC(I)+0>
        WRITSCOPE "J"                   ;LOAD ALL STIMULI INTO HARDWARE
        I=ISI<BLK>-DUR-38.              ;CORRECTED ISI <EMPIRICALLY DERIVED>
        WAIT I                          ;WAIT CORRECTED ISI.
        WRITSCOPE "S"                   ;TURN ON REFRESH.
        WAIT DUR                        ;WAIT STIMULUS DURATION
TRL1:   WRITSCOPE "KP"
WWW:    IF <RESP .EQ. 0>                ;WAIT FOREVER TILL RESPONSE
          THEN  WAIT 20.                ;NO RESPONSE YET.
                GOTO WWW.
          ELSE                          ;RESPONSE, GO ON
          DONE
FEED:   IF<COND<TRL> .EQ  0>                    ;PROCESS RESPONSE.
          THEN  IF <RESP .EQ. 1>
                  THEN  WRITSCOPE FD. "OK"      ;HIT
                        PC<0>=PC<0>+1           ;INCREMENT HIT COUNT.
                  ELSE  WRITSCOPE FD  "NO"      ;MISS
                  DONE
          ELSE  IF <RESP .EQ. 200>
                  THEN  WRITSCOPE FD  "OK"      ;CORRECT REJECTION
                  ELSE  WRITSCOPE FD  "NO"      ;FALSE ALARM
                        PC(1)=PC(1)+1           ;INCREMENT FA COUNT.
                  DONE
          DONE
        WAIT 500.
        WRITSCOPE "KP"
        RETURN
;------------------------  FORMAT STATEMENTS  ------------------------
FD      FORMAT<Z<200>L<14 >X<14 >A<0>>
        END
```

Figure 2. A sample experiment illustrating a variable-mapping multiple-frame version of the visual search task.

position over separate visual arrays as well as within an array. Consequently, there are three types of counterbalancing: The target's frequency, its temporal position, and its spatial position are all specified in the program.

Display generation, the second function, is also more elaborate than in the previous example. Each trial now consists of two displays, a study display and a multiple-frame test display. The major problem is the frame interval timing in the test display. Because timing is a general problem, it is described further in a later section on PLE performance characteristics.

The third function is response collection. Responses are collected through RECDK, which allows continued program execution while recording the response. Because RECDK does not interrupt program execution, the response variable (RESP) must be checked frequently for the occurrence of a response. Once a response is detected, the display sequence is terminated and feedback is provided to the subjects. Both the feedback and the display sequence termination are examples of displays made contingent upon responses.

The visual search example makes particularly clear

the potential of PLE for block-structure programming. The subroutine calls (e.g., CALL BALANCE) and the ALGOL-like IF-THEN-ELSE logic allow a program to be broken easily into subunits, facilitating both writing and debugging. Also, all of the input is handled in a single subroutine (INIT), as are the counterbalancing (BALANCE), stimulus presentation, and data collection (TRIAL). Like a tachistoscope, the program can accept stimuli ranging from digits to letter strings to graphic figures.

PERFORMANCE CHARACTERISTICS

Four characteristics are fundamental to PLE's performance as a tachistoscope: the temporal response of the system, including CRT offset timing, CRT onset timing, and interval timing; the spatial properties of the stimulus display, especially the spatial resolution and capacity; the intensity of the stimulus display; and the ability to define and present multiple fields. Each of these characteristics will be discussed below.

Temporal Properties

CRT offset timing. The offset timing of a CRT is determined entirely by the refresh interval and the phosphor decay time. The refresh interval of the IKON scope controller is 10 msec, with the refreshing of individual points occurring at 5-microsec intervals. The P-11 phosphor decays approximately exponentially to 10% of its original intensity in 80 microsec and .1% in 20 msec. To calculate the offset function entirely, one must know when the last refresh occurred relative to the offset. The IKON system makes this easy, because it always initializes a refresh clock at the beginning of each display. This gives one the ability to calculate exactly the number of refreshes in a display. For example, the 12-msec duration in the visual search experiment gives time for exactly two refreshes of the stimuli.

CRT onset timing. The onset delay of the CRT displays is a joint result of delays in PLE software and the IKON scope controller. Specifically, there is a fixed 10-msec hardware delay and a software delay that varies from 2 to 20 msec as the number of output characters varies from 1 to 120. However, the uncertainty of onset is very small, because the CRT refresh is begun only at the conclusion of the display generation. From this time, the only delay is the point-by-point display rate of 200 kHz. (The average character is 20 points and takes 100 microsec to display.) This allows millisecond timing of display durations and interdisplay intervals despite a fixed timing error.

Interval timing. Errors in interval timing come from delays due to intervening PLE statements and conflicts between operations at multiple stations. The former type of error is fixed for each instance and can be measured and corrected for when critical. The visual search example gives an illustration of this problem. Since the selection of individual displays takes 24 msec and their generation another 13 msec, this constant of 37 msec is subtracted from the display intervals to provide a corrected interval good to the millisecond. The only drawback with this strategy is that it places a lower bound on the possible display intervals (SOAs). When the duration of the displays is added to the selection and generation times, SOA is 50 msec.

The timing problems caused by multiple stations can only be understood through the nature of PLE's multiprogramming. Unlike most timesharing systems, PLE does not use timeslicing (i.e., placing a fixed limit on the processing duration of any one station). Instead, PLE allows each station to continue executing code until a timing, TTY, or SCOPE I/O command is encountered. These commands all suspend a particular station's control of the computer and allow other stations a turn. This scheme works because over 99% of the time in most programs is spent waiting for the clock or I/O. If this is not true, as is the case in the visual search experiment, the system starts to slow down. The problem begins when one station's I/O is completed during a second station's processing. The size of this delay is related to the duration of whatever code is uninterrupted by timing and I/O commands. Because the visual search experiment has 37 msec of uninterrupted code, the maximum possible delay to other stations is the number of other active stations times 37. For the visual search experiment, this error is too large, and therefore, this experiment can be run on only a single station.

Response timing. PLE's response timing is handled completely apart from the interval timing discussed above. The reaction times returned by the WAITK, RECDK, and WAITR statements are accurate to the millisecond.

Spatial Properties

Characters and graphics are represented by point plotting on a 256 by 256 dot matrix. With the CRT display subtending 14 deg (10 cm at a distance of 40 cm), the interpoint distance is about 3 min of arc (.4 mm). Adjacent points form a nearly continuous line on the CRT. The capacity of the display is set by the memory and refresh rate of the IKON controller. Currently, it can refresh 2,000 points in 10 msec, allowing for the simultaneous display of approximately 120 alphanumeric characters.

Intensity

The CRT intensity is manually calibrated using a knob on each scope and not under program control.

Field Selection

The basic advantage of any computer-controlled tachistoscope is the availability of an essentially

unlimited number of display fields. These fields may be constructed from separate components that can be combined in an infinite number of different patterns, thereby providing the potential for diverse and complex stimuli that would be impossible in a manual tachistoscope. Stimuli may be constructed to move about the screen or to change over time, as illustrated in the memory scanning example. Additionally, display fields may be made contingent upon the subject's response. The example experiments illustrate this capability in their use of feedback and the early termination of display sequences.

REFERENCE NOTE

1. Burkhardt, K. J. *The evolution of a higher level language for an on-line minicomputer system*. Unpublished manuscript, University of Washington, 1976.

REFERENCES

BURKHARDT, K. J. EMPP: An extensible multiprogramming system for experimental psychology. *Behavior Research Methods & Instrumentation*, 1976, **8**, 239-244.

BURKHARDT, K. J., PALMER, J., & WADDINGTON, W. An intelligent experimental station controller for human experimental psychology. *Behavior Research Methods & Instrumentation*, 1976, **8**, 369-374.

SCHNEIDER, W., & SHIFFRIN, R. M. Controlled and automatic human information processing: I. Detection, search, and attention. *Psychological Review*, 1977, **84**, 1-66.

SPERLING, G., BUDIANSKY, J., SPIVAK, J. G., & JOHNSON, M. C. Extremely rapid visual search: The maximum rate of scanning letters for the presence of a numeral. *Science*, 1971, **174**, 307-311.

STERNBERG, S. High-speed scanning in human memory. *Science*, 1966, **153**, 652-654.