Behavior Research Methods, Instruments, & Computers
1990, 22 (6), 520-525

— COMPUTER TECHNOLOGY —

How to obtain near-millisecond precision on the IBM PC with visual stimuli not exceeding one printed line

JAN GABRIELSSON
Uppsala University, Uppsala, Sweden

and

ROBERT J. JARVELLA
University of Umeå, Umeå, Sweden

Techniques are described for using an IBM PC equipped with a mouse to investigate the reading of character sequences of maximally one display line. Solutions are given to problems that arise and derive from the PC's real-time clock, the slowness of the display, and registration of input during a test session. A program for using the techniques to study asynchronous perception of printed words is described, and a demonstration program is provided as an appendix.

Studies of printed word recognition in which latency measures are used as data are manifold in the psycholinguistic literature (see, e.g., Coltheart, 1987; Henderson, 1982). Most such work has been done in the last 15 years, with laboratory computers. Some of the programs written, including one we describe here, have been implemented on IBM-compatible PCs. These computers are very common, and there are cheaper as well as more expensive models available, most of which are sufficiently hardware-compatible for using such programs.

A general aim in presenting visual stimuli and collecting timed responses is that the job be done as accurately as possible (see, e.g., Reed, 1979). In the present paper, we point out some obstacles one should be aware of, if one sets out to do high-precision timing with a PC. Programs having this purpose tend to be very machine-dependent, as is the Mental Lexicon Experiment (MLE) program presented below. The kernel of the program running the experiment is written in assembly language to make the critical parts as fast as possible, and to be able to manipulate the machine on a low level.

There are three sources of difficulty in putting together a reaction time (RT) program for studying visual processes on the PC: the clock, the display, and the input device used.

The work reported here was supported by the Swedish Research Council for the Humanities and Social Sciences (HSFR) and the Swedish Board of Technical Development (STU). Correspondence concerning this paper should be addressed to Jan Gabrielsson, Department of Computer Science, Uppsala University, S-751 20 Uppsala, Sweden, or to Robert J. Jarvella, Department of Linguistics, University of Umeå, S-901 87 Umeå, Sweden.

The Clock

Unfortunately, the real-time clock that DOS provides is too imprecise for timing milliseconds (its resolution is only 55 msec). A technique for reprogramming the real-time clock has been illustrated in a number of recent papers (see Crosbie, 1989; Dlhopolsky, 1988, 1989; Emerson, 1988). This (almost) standard method for reading time on the PC in milliseconds is useful when the stimulus presentation and the recording of data are continuous. However, for experimental paradigms with discrete trials, there is a better technique available, used for timing assembly language instructions (Sheppard, 1987). The technique involves the use of the internal 8253 timer chip and obtains a resolution of 840 nsec, but it requires one to stop the timer to read the result (thus losing continuous time). In the MLE program described below, the events of interest are discrete events. Together with other sources of timing errors, we use the technique to gain a total resolution still exceeding 1 msec.

The Display

Computer displays are slow. Monitors (depending on their graphic standard) rewrite the screen 50-100 times per second. Using a 50-Hz screen, it will take anything up to 20 msec until information sent to the screen is written there. Thus, it is necessary to know when this moment occurs. Most graphic adapters used on the IBM PC have a status register that allows one to tell when the raster beam is in the upper left-hand corner of the screen, just about to start rewriting the screen page. Using this information, one can synchronize the timer with the screen. If the stimulus is, for example, centered on the screen,

one subtracts half the screen-rewrite time from the response. Synchronization of a PC's display with millisecond timer to control the location of a stimulus is also taken up by Dlhopolsky (1989). In paradigms that depend on quickly removing graphic information, afterglow on the screen is a further problem.

Recording Responses

As an input device, we use a mouse. The obvious alternative—to use the keyboard—should be viewed with caution. Intelligent keyboards tend to buffer or delay responses. Graves and Bradley (1987), for example, have reported variable delays averaging 18.4 msec for an IBM PC keyboard, and twice that long for a PC-compatible keyboard. Using the mouse, the size of corresponding error will be much smaller, and also constant.

A simple solution for recording responses is to let the program listen for input while in a short loop. For millisecond resolution, one must then poll for input every millisecond. That may be too time-consuming, however, if there is some other computing to do in the program. The solution in MLE is the internal 8259A PIC (Programmable Interrupt Controller), which is used to generate an interrupt when a signal is recorded at the COM port to which the mouse is connected. Unlike the keyboard case, the signal here will arise whenever a mouse button is pressed or the mouse is moved, without the response being buffered or otherwise delayed.[1] The interrupt routine associated with the generated interrupt reads the timer and stores away the result. The mouse is then disabled, so that the subject cannot press the button twice and in so doing destroy the first result. Notice that the above set-up is possible only by going directly to the hardware (avoiding the mouse software driver). Also, the ball inside the mouse should be removed when buttonpresses alone are desired, because otherwise any movement of the mouse will generate interrupts, stopping the clock. We illustrate this interrupt technique in the demonstration program in the Appendix.

A Program Implementing These Principles

The MLE program was written to study the recognition of printed words displayed with small stimulus-onset asynchronies (SOAs). An SOA paradigm has proven to be a satisfactory means for asking whether a printed sequence is recognized in a single step (as a whole), or in more than one step (as a series of parts). If a sequence is recognized in one step, withholding some (any) of its letters even briefly should delay the sequence's recognition. If it is recognized in several steps, withholding particular (noncritical) letters at first should not have this effect. This issue is relevant in the testing of models of the recognition of morphologically complex words (see Jarvella, Job, Sandström, & Schreuder, 1987).

The MLE program provides flexibility in the asynchronous presentation of single and multiword stimulus strings of up to 80 characters long, as well as convenience in collecting and analyzing data. To use MLE, one needs a PC-compatible computer running the DOS operating system with two double-sided disk drives or a hard disk; at least 256K of memory; an IBM MGA, CGA, EGA, or VGA graphics adapter; DOS 2.11 or later; and a Microsoft-compatible mouse (serial version).

On entering MLE, one is presented with a menu, with which one selects a prepared data input (stimulus) file and defines an experiment. One important item shown in the menu is the raster-time unit: the amount of time the computer running the program uses to rewrite the screen (ca. 10–20 msec, depending on the machine), calculated every time the program is run. Multiples of this unit define what time asynchronies can be used between different parts of a stimulus.

In a stimulus file used with MLE, one formats each item as a string consisting of four elements:

1. The division of the character sequence that is desired (e.g., *Ice/hockey/ player*).
2. The order of appearance desired for those pieces (e.g., [2, 1, 3] would mean first display *hockey*, then *Ice*, and then *player*).
3. The delays desired between the pieces (e.g., {2, 3} means add *Ice* to *hockey* after 2 raster-time units, and add *player* to *Ice hockey* after 3 more raster-time units.
4. The correct response (e.g., *L for left button correct).

Since experiments often include repeated measures based on a set of items of the same type, the possibility also exists for specifying (2)–(4) for sets of items in an init file, which the program always searches for when started, but can also be specified using the menu.

An init file, a stimulus file, a file for writing results in, and an experiment name are all that is needed to run an experimental session with MLE. However, there are also a number of options that one chooses explicitly or selects by default. Two choices concern the order of stimuli desired (random or fixed), and the absence or presence of feedback after each trial (response and RT recorded). Fixed order together with verbose mode is a convenient option when one is checking materials in setting up an experiment. A number of more implicit choices can be made under the menu's heading "settings": whether one wants a random and/or constant delay before each stimulus appears; the maximum time it should stay on; whether it should be centered; whether a fixation point is desired, and if so, what ASCII character should be used; how many raster-time units in advance of the stimulus the fixation point should be turned off (to avoid masking caused by afterglow); whether one wants 40 or 80 character lines; and whether there is any condition for which RT need not be stored (e.g., in a go/no-go decision task).

When a subject has been run, MLE calculates a set of summary statistics in the subject's result file, including means, standard deviations, minimums, maximums, and errors committed in any condition, and it lists each stimulus together with the response given and its latency. Extended statistics that have the same form but that combine the result files for all subjects who have been tested using the same data file are available on request; these

give as output condition means, *SD*s, and so forth, over subjects, and pooled data for each item.

Program Availability

The MLE program and documentation for using it are available by contacting JARVELLA@SEUMDC51.bitnet.

REFERENCES

COLTHEART, M. (ED.) (1987). *Attention and performance XII: The psychology of reading*. Hove, U.K.: Erlbaum.

CROSBIE, J. (1989). A simple Turbo Pascal 4.0 program for millisecond timing on the IBM PC/XT/AT. *Behavior Research Methods, Instruments, & Computers*, **21**, 408-413.

DLHOPOLSKY, J. G. (1988). C language functions for millisecond timing on the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 560-565.

DLHOPOLSKY, J. G. (1989). Synchronizing stimulus displays with millisecond timer software for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **21**, 441-446.

EMERSON, P. L. (1988). Using serial interfaces and the C language for real-time experiments. *Behavior Research Methods, Instruments, & Computers*, **20**, 330-336.

GRAVES, R., & BRADLEY, R. (1987). Millisecond interval timer and auditory reaction time programs for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **19**, 30-35.

HENDERSON, L. (1982). *Orthography and word recognition in reading*. London: Academic Press.

JARVELLA, R. J., JOB, R., SANDSTRÖM, G., & SCHREUDER, R. (1987). Morphological constraints on word recognition. In A. Allport, D. MacKay, W. Prinz, & E. Scheerer (Eds.), *Language perception and production* (pp. 245-262). London: Academic Press.

REED, A. V. (1979). Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, **11**, 572-576.

SHEPPARD, B. (1987). High-performance software analysis on the IBM PC. *Byte*, **12**, 157-164.

NOTE

1. The interrupt is actually generated when a complete byte is read from the serial port. For instance, if the baud rate is 2,400, a constant delay of about 4 msec will arise. This can, however, be calculated to any desired level of precision.

APPENDIX

```
/*  Timing routine using a mouse as input device.
    Written in Turbo C 2.0
    Assumptions:
    -CGA (MDA) compatible graphic adapter.
     EGA and VGA will work as CGA if set to color text mode.
    -Mouse, hardware compatible with Microsoft's Mouse.
    -Mouse connected to COM port 1.*/

#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <mem.h>

#define MAX_COUNT           65535
#define COUNT_CONVERT       838.096    /* Each tick is 838.096 nsec */
#define TIMER_CONVERT       54.925     /* Each DOS tick is 54.925 msec */
#define TIMER_MODE          0x43       /* Timer control registers */
#define TIMER0              0x40
#define timer_low           ((unsigned far *)0x0000046C)
#define PIC_MASK 0x21       /* Interrupt controller mask */
#define PIC_EOI 0x20        /* Interrupt controller address */
/*#define VTRACE 0x03BA     /* Videosync address for MDA */
#define VTRACE 0x03DA       /* Videosync address for CGA */
#define NORM_ATTR 0x07      /* Normal video attribute */
/*#define SCREEN_BASE 0xB000 /* Base address for MDA screen memory */
#define SCREEN_BASE 0xB800  /* Base address for CGA screen memory */
#define INT_VECT 0x0C       /* Interrupt vector for COM 1 */
#define INT_DATA 0x03F8     /*  */
#define INT_DISABLE 0x10    /* Disable mask for COM 1 */
```

APPENDIX (Continued)

```
#define INT_ENABLE 0xEF       /* Enable mask for COM 1 */
#define V_RETRACE 1           /* Vertical retrace is about 1 ms */

#define FALSE 0
#define TRUE 1

unsigned ch, cl, new_timer, old_timer;
double timer_adjust,rt;
void interrupt (far *old_mouse)();
int key, mouse_running;

char *(keys[]) = {"Mouse moved", "Right key", "Left key"};

#define start_timer() \
     (outportb(TIMER_MODE,0x34), \
      outportb(TIMER0,0), \
      outportb(TIMER0,0), \
      old_timer = *timer_low)

#define stop_timer() \
     (disable(), \
      outportb(TIMER_MODE,0), \
      cl = inportb(TIMER0), \
      ch = inportb(TIMER0), \
      new_timer = *timer_low, \
      enable())

/* Given 840ns counter and 55msec counter calculate milliseconds */
double calc_time()
{    unsigned count;
     double usec, msec;
     count = MAX_COUNT-((ch << 8) | cl); /* Counter ticks backwards */
     usec = (double)count*COUNT_CONVERT/1000.0;
     msec = (double)(new_timer-old_timer)*TIMER_CONVERT;
     if (floor(msec) == 0.0 && usec < timer_adjust) usec = timer_adjust;
     return msec+(usec - timer_adjust)/1000.0;
}

#define disable_mouse() \
     (outportb(PIC_MASK,(inportb(PIC_MASK) | INT_DISABLE)), \
      mouse_running = FALSE)


#define enable_mouse() \
     (outportb(PIC_MASK,(inportb(PIC_MASK) & INT_ENABLE)), \
      mouse_running = TRUE)

#define eoi() outportb(PIC_EOI,PIC_EOI)

#define raster_on() (inportb(VTRACE) & 0x08)
```

APPENDIX (Continued)

```
#define sync_screen() \
    while(raster_on()); \
    while(!raster_on())

/* Mouse interrupt routine. When the subject presses a key the key pressed and
   time elapsed since start_timer was called is read. Mouse_running is set to
   false so the subject can't press the mouse again and destroy the data
   before we have a chance to read them. */
void interrupt new_mouse()
{
    if (!mouse_running) {
        eoi();
        return;
    }
    disable_mouse();
    stop_timer();
    key = inportb(INT_DATA);
    rt = calc_time();
    eoi();
}

/* Install the interrupt routine that will be called every time a mouse key is
   pressed or the mouse is moved.*/
void trap_mouse()
{
    eoi();
    disable_mouse();
    old_mouse = getvect(INT_VECT);
    setvect(INT_VECT,new_mouse);
}

/* Restore old mouse interrupt routine */
void free_mouse()
{
    setvect(INT_VECT,old_mouse);
    enable_mouse();
}

/* A character in the video buffer is made up of two bytes; the ascii code for
   the character and the video attribute. Every character is paired with a
   video attribute. The length of the new "string" and the offset into the
   video buffer according the given x and y are calculated.*/
char *prepare_string(str,x,y,pos,len)
char *str; int x, y, *pos, *len;
{   static char p[255], *s = p;
    while(*str) { *s++ = *str++; *s++ = NORM_ATTR; }

    *len = s-p;
    *pos = (80*y+x)*2;
    return p;
```

APPENDIX (Continued)

```
/* Transform the first byte that the MicroSoft mouse returns into 1 for left
   key and 2 for right key. 0 means that the mouse just moved. */
#define key_trans(A) ((A & 0x30) >> 4)

/* Move string to video buffer */
#define draw_string(S,OFFS,LEN) \
    movedata(_DS,(unsigned)S,SCREEN_BASE,OFFS,(size_t)LEN)

main()
{   double d,t;
    char *st;
    int pos,len;

    /* Calibrate timer */
    timer_adjust = 0;
    start_timer();          /* Start and stop timer to get overhead */
    stop_timer();
    t = calc_time();        /* Time (in usec) to time "nothing" */
    timer_adjust = t * 1000.0;
    clrscr();

    /* Time how often the screen is redrawn */
    sync_screen();          /* Wait for beam to be in upper left corner */
    start_timer();          /* Start timer */
    sync_screen()           /* Wait for beam to be in upper left corner */
    stop_timer();           /* Stop timer and read time elapsed */
    t = calc_time();
    printf("Screen is redrawn every %.011fms\nRETURN to continue...",t);
    getch();

    /* A small experiment... */
    st = prepare_string("Syncronizing",20,10,&pos,&len);
    trap_mouse();           /* Set up mouse to call our interrupt routine */
    sync_screen()           /* Wait for beam to be in upper left corner */
    enable_mouse();
    start_timer();          /* Start timer */
    draw_string(st,pos,len); /* Move our string into the video buffer */

    while(mouse_running);/* Wait for subject to press a key */

    free_mouse();       /* Restore mouse interrupt routine */

    /* The string appeared in the middle of the screen, so half the time it
       takes to redraw the screen is subtracted from the response time. */
    rt = rt - (t-V_RETRACE)/2.0;
    printf("RT %.011fms, %s",rt,keys[key_trans(key)]);
    getch();
}
```

(Manuscript received May 25, 1990;
revision accepted for publication August 30, 1990.)