Behavior Research Methods, Instruments, & Computers
2003, 35 (1), 116-124

DMDX: A Windows display program with millisecond accuracy

KENNETH I. FORSTER and JONATHAN C. FORSTER
University of Arizona, Tucson, Arizona

DMDX is a Windows-based program designed primarily for language-processing experiments. It uses the features of Pentium class CPUs and the library routines provided in DirectX to provide accurate timing and synchronization of visual and audio output. A brief overview of the design of the program is provided, together with the results of tests of the accuracy of timing. The Web site for downloading the software is given, but the source code is not available.

DMDX is a Win32 program designed to precisely time the presentation of text, audio, graphical, and video material and to enable the measurement of reaction times (RTs) to these displays with millisecond accuracy. It represents an extension of a suite of DOS-based programs known as DMASTR, developed and tested at Monash University in Australia over a period of 15 years, starting in 1975, by a team including the first author, Rod Dickinson, Wayne Murray, and Mike Durham. Graphics and sound capabilities were added in 1989 by Jonathan Forster, and the extension to a Windows 9x platform was carried out by Jonathan Forster at the University of Arizona in 1997.

The purpose of the present article is (1) to inform the research community about the existence of the software and its capabilities, (2) to provide a nontechnical explanation of how the software works, and (3) to answer the skepticism expressed in some quarters about the possibility of using the Windows operating system in a real-time environment (see, e.g., Myors, 1999).

First, a brief historical note. The DMASTR suite was originally developed by the first author in 1975 for a PDP-11 computer running under RT-11. It was written in assembler code (MACRO) and was an interrupt-driven program that synchronized the activity of the display program with the position of the raster in the display monitor, thereby enabling accurate measurement of the time interval between when the display actually appeared and when the subject responded to that display. The RT-11 operating system gave the programmer total control over all the operations of the computer, so that the programmer knew precisely when each display event occurred and when each response was made by the subject. With the advent of far cheaper IBM-compatible PCs, the switch was made in 1983 to a DOS-based program (DM) written in C with no loss of control over timing. This version of DMASTR was still purely text based. In 1989, the second author extended the system to include graphical displays and sound, using the Borland Turbographics C library. However, this program (DMTG) was restricted to a particular graphics format (not widely supported), a particular speech-editing system (BLISS, developed by J. Mertus at Brown University), and as a consequence, it was also limited to a narrow range of sound cards for which BLISS drivers had been written.

As MacInnes and Taylor (2001) point out, attempting to stay with DOS means that researchers are restricted to outdated hardware and software, a problem that becomes more acute with each passing year. As the supply of sound cards suitable for DMTG began to dwindle (suppliers going out of business or completely changing the design of the card), it became clear that any new application having a life span of more than a year or two would have to be based on the de facto Windows standard. The idea was that if a piece of hardware worked with Windows, it would also work with DMASTR. In that way, we could keep abreast of new technology without having to write new device drivers for every new piece of hardware to come on the market. That was the goal that drove the development of DMDX. In addition, there was the obvious benefit of greatly enhanced graphics. Furthermore, the fact that DMDX was a Windows program meant that it used accepted formats for fonts, image files, and sound files, which in turn gave the user a wide range of support tools.

DMDX is a hybrid name. The DM in the name indicates its lineage as part of the DMASTR system. The DX refers to DirectX, a set of DLLs (dynamic link library routines)

Over the years, a number of individuals have made invaluable contributions to the development of the DMASTR system. They include the late Rod Dickinson (to whose memory this paper is dedicated), Mike Durham, Wayne Murray, and Kevin Ackley. The support of the Psychology Department at the University of Arizona has also been critical, as have the criticisms, suggestions, complaints, and requests of many colleagues who have used DMDX. Special thanks go to Matt Davis and Mike Ford (MRC Cognition and Brain Sciences Unit) and Anna Woollams (Macquarie University). Finally, the senior author expresses his profound gratitude to the junior author for creating such a professional extension to the DMASTR family. Correspondence concerning this article should be addressed to K. I. Forster, Department of Psychology, University of Arizona, Tucson, AZ 85721 (e-mail: kforster@u.arizona. edu).

that gives the Windows programmer access to the actual hardware—a development driven largely by the need to provide a fast-action, dynamic gaming experience. When DMDX requires something to be displayed, it issues a command to DirectX, not to Windows.

DMDX runs on Win32 machines only (this rules out Windows 3.1), and DirectX should be available as part of the system.[1] Details of the earlier DOS programs (DM and DMTG) and the current version of DMDX can be found at the DMASTR Web site (http://www.u.arizona.edu/~kforster/dmastr/dmastr.htm). Documentation for DirectX can be obtained by downloading the DirectX Development Kit from http://www.microsoft.com/directx/download.asp. However, users of DMDX do not require an understanding of DirectX.

Timing Problems With a Multitasking Operating System

The problem with asking Windows to do something at a specified time is that Windows is a multitasking environment. There may be other applications running that have a higher priority, and hence, the operating system will not always carry out an operation whenever it is requested. Even if there are no other applications running (something that we insist on for accurate timing in DMDX), the Windows kernel may still interrupt the execution of DMDX to determine whether any other task needs to be serviced. These tasks may include such things as networking, disk maintenance, and so forth. Also, DMDX consists of a number of subprograms (*threads*) that carry out different tasks. One thread may be involved in loading display material into the video RAM, another may be involved in playing a sound file, while a third may be monitoring the position of the raster on the video monitor. All of these processes may be operating at the same time.

The most obvious situation in which this presents a problem is in the synchronization of the display routines with the video raster. This is critical for accurate measurement of RT, since the software can measure the RT only from the time that the appropriate page in the video RAM is activated, not when the information in that page actually becomes visible on the monitor screen. If new material is loaded into a page regardless of the position of the raster, it is possible that this material will not be displayed until the next refresh cycle. To achieve the required synchronization, the register that indicates the status of the raster has to be monitored constantly. The signal indicating that a refresh cycle has been completed is very brief and is easily missed if the program diverts its attention from this task, even momentarily. To avoid this, it would be necessary to constantly monitor the raster position, which would require a very tight loop, so tight that the program would be unable to carry out any other function. Even so, the Windows kernel regularly interrupts the current application for a period longer than the refresh signal, so it is inevitable that some refresh signals will be missed. To get around this problem, DMDX is provided with information about the refresh cycle time by a sister program, TimeDX. This program times the refresh cycle and stores the value in the registry, so that DMDX can later retrieve this information. If, for example, the refresh cycle is specified as 16.666 msec, DMDX knows that it can ignore the raster position if the time since the last refresh cycle is less than, say, 13 msec. So the thread that is responsible for detecting the retrace signal can be put to sleep for 13 msec (allowing other threads to operate), but once 13 msec has elapsed since the last refresh, the retrace thread wakes up and resumes the task of constant monitoring until the refresh signal is detected.

But a problem arises if some other process with a higher priority than the retrace thread is required during this critical phase. If such an interrupt disables the thread just as the refresh signal is about to occur, the refresh signal will be missed. This would mean that DMDX would lose track of 16.6 msec of display time and the current stimulus would be displayed for at least one extra refresh cycle. For many applications, this might not matter very much, but for time-critical procedures such as priming paradigms with very short prime durations, this error might be intolerable. Of course, the problem could be much worse, since if the probability of missing a refresh signal is reasonably high, DMDX could lose track of multiple refresh cycles.

The second situation in which a timing problem could arise is when the subject makes a response. DMDX polls the state of the input devices every millisecond, and if a response is indicated, the time is recorded. If the thread that is responsible for carrying out this task is momentarily preempted, the time will not be read until that thread regains control, and the RT will be overestimated.

Solutions to the Timing Problems

Display timing errors. The solution to the problem of timing the display is provided by the high-performance timer, a piece of hardware on all Pentium-class motherboards. This device keeps track of elapsed time down to submicrosecond accuracy (1.19 MHz) and operates autonomously, regardless of what application is currently running. If DMDX fails to detect a refresh signal, DMDX will discover the error when it notes that, according to the high-performance timer, more than 16.666 msec (or whatever the refresh rate actually is) have elapsed since the last refresh. DMDX then simply assumes that a refresh must have occurred and adjusts its counter accordingly.

This process could be repeated indefinitely. For example, if DMDX has been told that the display must be changed after 200 ticks (a tick is the time taken for one refresh cycle), it could theoretically ignore the raster for 199 ticks, using the high-performance timer to keep track of the elapsed time. However, since the clock speed for the timer and the raster might differ slightly, a slight drift could occur over such a long time period. To avoid this problem, DMDX resynchronizes itself with the raster whenever possible, reducing drift to negligible amounts.

This procedure copes with all errors except one, and that is when an error occurs on the *final* refresh cycle. For example, suppose that a target word is to be displayed for three ticks but, on the third cycle, the refresh signal is missed. This means that the display of this stimulus will not be terminated until the next refresh cycle, which means that the duration of the stimulus will be one tick longer than it should have been.

There is no way to recover from such an error. However, in our experience, it is extremely unlikely, given adequate video card drivers. The reason is that DMDX queues display buffers (up to 24 frames), so that the material for the next frame is already loaded into the appropriate video page. All that is needed for correct operation is that a request to switch to the next video page be issued at any point during the current refresh cycle. The switch then takes place automatically when the refresh cycle is completed. In order for an error of this type to occur, the display thread would have to be preempted for an entire refresh cycle. Given a reasonably fast CPU, this is not going to happen very often. On the fairly slow test machines used in our laboratory (166-MHz Pentium MMXs), the probability of such an error appears to be very small.

The probability of an error's occurring can be determined by running the tests provided in TimeDX. However, knowing that a display timing error is likely to be rare is reassuring, but one might still want to know whether an error occurred during an actual experiment and, more important, *when* it occurred. This information is provided to the user in the output file. If a given frame is displayed later than it should have been, the item number of the trial and the frame number are indicated, along with the size of the error. So even if errors do occur, the user can discard the data from those trials. In our experience, such errors are extremely rare when DMDX is run as the only application. We have scanned the output from our laboratory machines for more than 100,000 time-critical trials involving masked priming (where the precise duration of the prime was absolutely critical) and have found only two examples where a frame was displayed longer than scheduled. In one case, the error was in a noncritical frame, and in the other, the critical frame was displayed for one tick longer than specified. However, on an office desktop with multiple applications loaded (e.g., antivirus software, alarms, calendars, backups, e-mail, etc.), display errors can be quite common.

The major cause of display errors appears to be the quality of the video drivers supplied with the graphics card, rather than the CPU speed. Very often, display errors can be corrected simply by downloading the latest drivers. However, with a moderately fast CPU (500 MHz), one can virtually guarantee error-free displays.

Response timing errors. As was mentioned above, measurements of RT may be subject to similar problems. The nature of the problem will depend on the input device. DMDX supports a parallel I/O card (PIO12), a joystick or gamepad (connected via the game port or a USB port), a mouse, a keyboard, or any other input device that has DirectX drivers.

DMDX polls the status of the PIO12 and the joystick every millisecond, but this operation depends on the multimedia timer callback originating from the Windows kernel. DMDX requests the kernel to call it every millisecond, but in actual fact, the interval between successive callbacks varies, depending on what other tasks the kernel was attending to.[2] So if the callback immediately following the occurrence of the response is delayed, the RT will be overestimated. The severity of this error can be estimated by using the test routines provided in TimeDX (see below).

For the mouse and keyboard input, the nature of the problem is slightly different. The occurrence of mouse clicks and keypresses is signaled to DMDX by DirectX and, hence, is subject to the variation that introduces.

The only real solution to this problem would be to install independent hardware capable of detecting when a stimulus appears on the display screen and measuring accurately the time until a response occurs. Such a system has in fact been used by McKinney, MacCormac, and Welsh-Bohmer (1999). However, this requires the user to purchase additional hardware for each experimental station, and the cost–benefit ratio of this solution might be high. For example, an average error of ± 1.5 msec can probably be safely ignored in most RT experiments.

How DMDX Operates With DirectX

The input to DMDX is an .rtf file (rich text file), generated by applications such as Word or WordPad. This is referred to as the item file or script and specifies what is to be displayed and how it is to be displayed (see the examples below). The first line of this file (termed the *Parameter* line) specifies such things as which input devices are to be used, the video display mode to be used, the default exposure time for a frame, and so forth.

Preliminary preprocessing. Initially, DMDX parses the item file taking note of all RTF control codes. Next, if the item file calls for scrambling of items, the scrambling routines produce another file that contains a pseudorandom item sequence. Following these preliminary operations, the various processes that make up DMDX proper are begun. First DirectX is instructed to switch the screen to whatever display mode has been requested in the item file. That mode must have been set up with TimeDX beforehand, since DMDX reads the value of the refresh rate from the registry that TimeDX puts there. It also requests that as many screen buffers as can be contained in video memory be created. These are called DirectDraw Surfaces. Beside the primary surface that is always displayed, there can be a large number of "back" surfaces that DMDX uses for buffering (up to 24 on an 8M video card at 640 × 480 with 8 bits per pixel [bpp]). Following this, the millisecond callback from the Windows kernel is initiated. Next, a thread is created to keep track of the retrace. A thread is another task running in parallel with the original program (itself just another thread). The retrace thread spends most of the time sleeping (i.e., allowing other threads to execute). Once each retrace period, it wakes up and waits for a number of milliseconds until it either finds the retrace or decides that it has missed it; in either event, counters are updated, and DMDX then decides what should happen next. As was indicated earlier, in most cases, missing a retrace is not a problem, so long as DMDX gets control back before the next retrace.

A thread is also created for each input device specified in the item file (the default being the keyboard). Threads for keyboards and mice and any other interrupt-driven devices simply wait for input data, whereas all other polled devices have threads that are woken up periodically by the millisecond callback to check whether a key is being pressed. Another thread can exist to handle requests to play sound files; however, this thread is not created until DMDX finds a request to play a sound. At that time, the sound system as a whole is initiated, and the sound thread is created, which can take quite a number of milliseconds. If DMDX is going to be used primarily to present audio stimuli, the priority of the audio thread can be increased by the user.

Per-item processing. An item is presented when the subject requests it, either by pressing a key or by pressing a footpedal, and so forth. Prior to the request for an item, DMDX performs all disk-related tasks. The experimenter can specify the delay between the request and the first frame, but care must be taken to ensure that sufficient time is allowed for these tasks to be completed (DMDX provides an output indicating the time taken to prepare items). Disk-related tasks include writing diagnostics to disk, writing result files to disk, reading the item to be displayed (although this is likely to have been buffered in memory, since DMDX uses file mapping), and reading the bitmaps and sound files used in the item and storing them in the appropriate buffers. If a sound file is used for the first time, this is when the sound system is initiated.

Once the request has been received, DMDX prepares each individual frame in a separate DirectDraw surface (stored in main memory, rather than in screen memory). It then ascertains which screen surface that frame will be displayed in and what information needs to be preserved or erased from the previous display. Having done that, it creates a surface just big enough to contain the region that will change on the screen surface and copies the corresponding region of the memory surface to this new temporary surface. Once everything is drawn, DMDX schedules the frames for display and adds them to the display queue. This consists of a queue of frames, along with the time at which they must be displayed.

Per-frame processing. DMDX's main thread watches the display queue to see whether any of the screen surfaces become empty (or are already empty at the item's commencement) and moves the display queue surfaces onto the screen surfaces one segment at a time, in order to avoid locking out all other threads for a substantial period of time. When a video retrace is detected, the retrace thread examines the display queue to determine whether the next screen surface should be activated (*flipped*), when sounds should be commenced, when output bytes should be output, when the timing of the response should begin, and so forth. The timing of these requests is calculated ahead of time and is stored on another queue, which is examined by the millisecond callback thread. The operations controlled in this way include initiation of a sound sequence, output of information to a parallel I/O card, polling of input devices, calculation of RTs, and so forth.

Accuracy of Timing

In order to determine just how serious the timing errors might be, we have extensively tested the accuracy of DMDX with various input devices, and these data are reported in the next sections. Readers can judge for themselves whether the accuracy is sufficiently high to meet their needs.

Video synchronization. It is one thing to be told that DMDX is synchronized with the video raster, but another to be able to see that it is. Users can easily verify this for themselves by constructing a series of frames, each consisting of a column of Xs displayed one column to the right of the preceding frame. Each frame is displayed for some brief interval. If there is no synchronization, then occasionally, the display will be incomplete. For example, the first two columns might be complete, but only the first half of the third column is displayed, and the rest of the display is blank. With repeated viewing of the same sequence, it is not difficult to determine whether each column gets displayed. With perfect synchronization, this should always be the case.

Accuracy of refresh rate timing. TimeDX provides such a test. The test swaps the contents of the video page, so that the display alternates between a red background and a blue background. The swap is made regardless of the position of the raster (some versions of DirectX and some video card drivers do not permit such an operation). The timing of these swaps is controlled by the high-performance timer, the interval between the swaps being exactly half the retrace interval (i.e., at a rate twice that of the refresh rate). If TimeDX is using the correct value for the refresh rate, there will be perfect synchronization between the time at which the swap occurred and the position of the raster. The display will consist of two segments—the top segment would be red, the bottom would be blue, and the border between them would be stationary, with the exception of a few flickers as TimeDX gets preempted by other processes. If the buffers are swapped at a slightly faster rate than the refresh rate, the border will scroll slowly up the screen; if the buffers are swapped at a slower rate, the border will scroll down the screen. If the border is scrolling slowly, the user can adjust the swap rate until the border becomes stationary. This gives a new, more accurate measure of the refresh rate, which can then be stored in the registry.

Accuracy of display timing. In language-processing tasks, such as RSVP (Forster, 1970), a sentence is displayed one word at a time at a rapid rate. If DMDX cannot move the display queue surfaces onto the screen surfaces rapidly enough, a display error occurs, and one or more of the frames will be displayed for a longer period of time than specified (most likely, for one screen refresh), which would distort performance on that item. To assess DMDX's performance under these conditions, we selected an older Pentium (166 MHz) with 64 MB of RAM and a 4-MB S3 Virge video card, running under Windows 98. A phototransistor was placed at the top left corner of the screen. A 10-word sequence was then displayed in white letters on a black background at the middle of the screen, each word being displayed for two refresh cycles (for this setup, 27.46 msec). The video mode was 640 × 480 with 8 bpp. As the first word was displayed, the RT clock was turned on. At the conclusion of the sequence, a white mask was briefly displayed, which activated the phototransistor, which in turn triggered a response to the display, using the parallel I/O interface. If there were no display errors, each frame was displayed for exactly the number of refresh cycles specified by the item script, the determination of the refresh cycle time by TimeDX was accurate, and the calculation of the RT functioned perfectly, then the observed RT should be 10 × 27.46 = 274.6 msec, plus the time required for the phototransistor to trigger a response. If a display error occurred and a frame was displayed for three refresh cycles rather than two, the RT would be longer by a factor of 13.73 msec (one refresh cycle). If more than one display error occurred, the RT would be increased still further. However, over 100 tests, no display errors were reported by DMDX. The mean RT was 278.56 msec with a standard deviation (*SD*) of 0.53 msec (the range being 277.1–279.4 msec). This amounts to an error of less than 4 msec, most of which was due to the time taken for the phototransistor to stop the clock, with the remainder being due to an error in registering precisely when the response occurred (see the next section). Given the fairly demanding nature of the RSVP display requirements, this performance seems entirely adequate.

In the previous test, the video mode allowed DMDX to use 12 buffers, which means that the entire sequence could be assembled and buffered prior to the display. Switching to a 640 × 480 display with 16 bpp instead of 8 (double the memory requirements) restricted DMDX to only five buffers,[3] meaning that the entire sequence could not be assembled in advance. However, under these conditions, once again no critical display errors were reported, and performance was virtually unchanged, with a mean RT of 278.99 msec with an *SD* of 0.53 msec.

Response timing. Errors in response timing can arise from several sources, such as the physical properties of the input device itself (e.g., switch closure time) or the nature of the interface (e.g., serial vs. parallel input). As far as the software is concerned, the main source of error is a delay in the millisecond callback from the Windows kernel. If the hardware has registered that a response has occurred but the millisecond callback to DMDX has been delayed, the time at which the response occurred will be recorded incorrectly. To test the accuracy of response timing, we constructed test hardware that produced a electronic switch closure[4] every 524.3 msec (a 2.000-MHz crystal divided by 2 to the 20th power). This interval was chosen to approximate the length of the average lexical decision time, which is the most commonly used task in our laboratory. One useful feature of DMDX is that it includes a number of different modes of operation, designed to assist the user in evaluating the performance of his or her own installation. One of these test modes (Test Mode 8) simply records the time between successive response signals.

The machine tested was an AMD-K6 300 with a Riva 128 4M video card and 64 MB of RAM, running under Windows 98. Table 1 shows the frequency distributions of timed interresponse intervals obtained with various input devices under different conditions, using Test Mode 8. The first four tests listed in Table 1 were carried out with the recommended input device—namely, a parallel I/O card (a MetraByte PIO12).

In every case, the mean interresponse time was 524.3 msec. However, this is misleading, since any delayed callback is issued as soon as the kernel regains control. So, if one callback is delayed by 1.5 msec, the interresponse time will be overestimated by 1.5 msec, but the next will be underestimated by exactly the same amount. The more important values are the *SD*s. For the PIO12 input device tested without any simultaneous activity, the *SD* was 0.84 msec. In practical terms, this means that the measurement error was less than 1 msec on 75% of the trials and, on the remainder, was never greater than 2 msec. These figures were obtained using Version 1 of DMDX. A similar test carried out on the improved Version 2 of DMDX, using a Celeron 400 multimonitor system with a Riva TNT 16M primary display (for the experimenter) and an S3 Virge GX 4M secondary display (for the subject), produced an *SD* of 0.77 msec. Neither of the machines used for these tests had a CPU speed (300 and 400 MHz) that could be described as fast by today's standards.

Other tests were carried out under more stringent conditions. Table 1 shows the results when the same test was carried out while DMDX was playing a 11025-KHz 16-bit

Table 1
Frequency Distributions of Interresponse Times for DMDX Operating With Different Input Devices Under Varying Load Conditions (Running on an AMD-K6 300-MHz Machine Under Windows 98)

	Observed Time Interval (msec)									
Input Device	508–521	522	523	524	525	526	527	530	*N*	*SD*
PIO12 (no load)	0	0	87	200	239	46	0	0	572	0.84
PIO12 with sound	0	2	90	193	227	50	0	0	562	0.87
PIO12 with graphics load (30 × 250 × 8 bpp)	0	12	81	170	180	59	7	0	509	1.01
PIO12 with network load	0	0	92	196	230	53	0	0	571	0.86
Mouse (no load)	0	0	3	334	222	12	0	0	571	0.55
Keyboard (no load)	89	0	7	1	0	152	321	1	571	5.15

.wav file lasting approximately 3 sec. The *SD* in this situation was 0.87 msec, virtually the same as that without the sound file. This indicates that the demands imposed by playing a sound file have very little impact on accuracy of response timing.

A more stringent test was to measure accuracy under unusually severe conditions. Normally, tachistoscopic experiments involve a brief display of a single image, followed by some type of mask and then nothing. A more demanding situation is one in which an extended series of images must be displayed in rapid succession—for example, a different display every 100 msec. This test produced an increase in the variability (*SD* = 1.01 msec). However, the maximum error never exceeded 3 msec.

Since the machines running DMDX are likely to be connected to a local Ethernet network, one might ask whether the amount of traffic on the network is likely to interfere with the measurement of RTs. This was tested with an ISA 3Com Etherlink III card by transferring a 500-MB file between two other computers over the network, producing a constant 5-Mbps load while the timing test was carried out. There was again no detectable effect (*SD* = 0.86 msec). It should be noted that 5 Mbps is an extremely large amount of traffic.

Other input devices were also tested. For example, a Microsoft Serial Mouse 2.0A (with the ball removed) produced an *SD* of only 0.55 msec. The keyboard proved to be far more variable, with the *SD* jumping to 5.15 msec. The maximum error was a huge 16 msec, but this occurred only once out of 571 trials. However, it should be noted that this variability has nothing to do with the software or the operating system but depends on the circuitry involved in the particular keyboard. Some keyboards poll the keys at a faster rate than others and may well produce better results. Hence, the results listed here must be treated as illustrative only.

Measuring variability in absolute RTs. Another method of testing involved triggering a MetraByte PIO12 input card with a phototransistor focused on the monitor display. In this test, the response occurs immediately after the stimulus becomes visible (to the phototransistor). Any variation in the observed latency must, therefore, reflect variation in the callback routines from the Windows kernel. Over 100 trials, this procedure produced a standard deviation of 0.29 msec. This means that 95% of the observations fell within an interval of 1.16 msec, which is almost exactly millisecond accuracy.

The *SD* gives a measure of variability but says nothing about the constant error. Does DMDX overestimate the true RT and, if so, by how much? There are two points to make here. First, the absolute RT is rarely the object of interest. Usually, we are interested in differences across conditions, and in this case, the constant error is irrelevant. Second, it is difficult to get a measure of the "true" RT—that is, one that is independent of the physical equipment used to measure it. For example, we can focus a photocell on the screen (as in the previous example) that fires a 12-V solenoid as soon as the display is detected, which in turn depresses a switch of some kind. Ideally, the observed RT would be close to zero. However, with the solenoid depressing a PIO12 microswitch, the RTs ranged from 18 to 20 msec. But without knowing how long it takes the photocell or the solenoid to respond or how long it takes for the key to be pushed its full travel distance, we cannot really interpret this figure. All we know is that the constant error cannot be larger than this figure but is likely to be very much smaller. However, we can at least rank input devices according to the size of this error. The values obtained are shown in Table 2.

The clear indication from the data in Table 2 is that one needs to be very careful about using a keyboard, since obviously there is considerable variability from one device to another. On the other hand, the data for the particular MS Serial mouse that we tested were surprisingly good. Although the RTs might be overestimated, the variability was quite small and did not vary as a function of whether the ball was removed or not. However, it should be stressed that this may be true of only this particular mouse. There may be considerable variation in performance across different manufacturers, or even within a given model. Finally, by far the best performance is produced by the PIO card, although a conventional joystick or gamepad comes very close.

Variation in callback latencies across operating systems. As was mentioned earlier, one source of error in response timing is the fact that the Windows kernel does not always activate DMDX every millisecond. TimeDX provides a quick method of measuring variation in the

Table 2
Minimum and Maximum Response Times
With Different Input Devices

Device	Response Times (msec)	Comment
PIO12 microswitch	18–20	Baseline (involves smallest amount of travel)
PIO12 KB switch	31–33	Longer values due extra 6 mm of travel
Generic joystick (also gamepad)	28–31	Polled at the default rate (every 3 msec)
MS serial mouse (without ball)	44–50	
MS serial mouse (with ball)	46–52	
Old AT keyboard	40–47	Note variation across different keyboards
OmniKey 102 KB	33–40	
Cheap Win95 KB	33–69	

callback latency. This is not a perfect indicator of the *SD* while DMDX is executing, since TimeDX is not playing audio files, polling input devices, or keeping track of the position of the raster, although it is involved in fairly intensive screen output. Nevertheless, it provides a relative measure that is suitable for comparing different CPUs and different operating systems. Under Windows 98, the test machines used to generate the data in Table 1 produced a callback *SD* of 0.29 msec. Under Windows 2000, this variability was reduced to only 0.07 msec. Under Windows ME and XP, the *SD* was 0.08 msec. It is worth noting that initially, the disadvantage of Windows 2000 and XP was that the PIO card could not be used as an input device (or as an output device), because no drivers for this device were available. However, this is no longer the case.

Conclusion. Considering the amount of variation in the time required for even simple cognitive operations, where *SD*s for a given individual in excess of 100 msec are not uncommon, it appears that the amount of noise introduced by timing errors in DMDX are relatively minor. However, it must be stressed that the figures reported in Tables 1 and 2 are valid only for the particular configurations used in our laboratory. Faster CPUs and more efficient graphics cards will reduce the variability still further.

Why Is a Win32 System Thought to be Inadequate?

If timing problems are so effectively coped with, one might ask why commentators such as Myors (1999) have asserted that Windows is totally inadequate for precise timing. For example, Myors used the keyboard autorepeat as a signal generator, rather than an external response device, as we have, and ran a program written to time the interval between received keystrokes. When this program was running in a DOS window under Windows 95, it was incapable of detecting the video retrace and yielded a distribution of times with an *SD* of 16.0 msec, whereas the same program running in MS-DOS mode (when Windows was not active) produced an *SD* of only 0.06 msec. Clearly, Myors's test yielded an unacceptably high *SD* for the Windows test, whereas our tests did not.[5]

Why should there be such a difference? The answer is that the example Myors used was a program written for DOS. He found that this was far more accurate when it is run under DOS than under Windows. What it should have been compared with is a comparable program written for Windows. DMDX is a program written specifically to take advantage of the functionality offered by Win32, whereas Myors' program ignored it.

The Capabilities of DMDX

System requirements. To run at all, DMDX requires a Pentium PC running Windows 95, 98, 98SE, ME, XP, or Windows 2000, with at least DirectX 5 installed, with DirectX 7 providing some enhanced timing (DirectX versions beyond 3 are not permitted with Windows NT). A CPU speed of 500 MHz is ideal, but not essential, since the lab machines at the University of Arizona run perfectly well at 166 MHz. To run effectively, 64 MB of RAM are sufficient, with a 4-MB video card and a sound card. It should be noted that a parallel I/O card can now be used with Windows 2000 or XP, which is essential for the coordination of the operation of DMDX with external devices, such as fMRI scanners and so forth.

Operating modes. DMDX runs in several different modes. The standard mode is a simple RT situation, in which the subject makes a binary classification of the input. There are three inputs: a positive response, a negative response, and a request. The request initiates a new trial and is used for self-paced experiments. These inputs can be interfaced with a parallel I/O card, a mouse, a game pad (or joystick), or the keyboard (with the concomitant cost to RT accuracy).

In addition, there is a mode in which the subject can make multiple responses over time. DMDX records the nature of each response and the time at which it occurred. This mode is normally used with keyboard input, which makes it possible to record ratings, or typed responses. Typed responses can be echoed to the display screen.

If a parallel I/O card is installed, DMDX can also be programmed to output signals to control external devices or to provide timing signals.

DMDX supports a dual-monitor display mode, in which the experimenter views a separate display from the subject. Single-monitor mode is also supported. In addition, it is possible to track the course of an experiment from a remote location, since DMDX can be programmed to send information about each trial as it occurs to a user-specified Internet address.

DMDX also provides a digital VOX mode, designed for recording vocal onset latencies. This eliminates the need for an external device. Users can specify the threshold intensity required to trigger the VOX and have the option of obtaining a .wav file output for each trial, indicating the point at which the VOX was triggered, so that after the testing session, the experimenter can review the appropriateness of the trigger point and adjust the measured RT accordingly.

Types of display. DMDX supports standard Windows graphics, which includes Windows fonts and .bmp and .jpg files. Sound files (.wav files) can be played simultaneously with graphical displays, allowing for cross-modal experiments. Considerable effort has gone into the procedure for synchronizing visual probes with audio files. Support for the display of digital video files (.mpg, .mov, or .avi) is also provided. Users can measure RTs to critical frames within these files, although the timing here is more subject to variability.

Experimental scripts. The experiment is controlled by a script written in rich text format (.rtf), using Microsoft Word or WordPad. The first line of the script sets a number of parameters, such as the default frame duration, whether the items are self-paced, how the order of the items should be scrambled, and so forth. The specifications for each item then follow. A typical item in a lexical decision experiment testing for semantic priming might look like this:

+001 "+" %70 / "doctor" %35 / * "NURSE" %70 / ;

The plus sign at the beginning of the item indicates that the correct response is a positive (*yes*) response (i.e., the target, "NURSE," is a word). Everything between quotation marks is displayed on the screen, by default centered both vertically and horizontally. The display sequence is divided into a series of frames, and the "/" symbol functions as a frame delimiter. In this example, when a request for a display is received, the first frame displaying a fixation point ("+") is presented. The duration of this frame is determined by the frame timer "%70," which specifies that the next frame onset should be delayed by 70 video ticks. For a monitor refreshing at 70 Hz, this would be 1,000 msec. The next frame presents the word "doctor" for 35 ticks, and this is then followed by the frame presenting the target, "NURSE." Simultaneous with this frame onset, the RT clock is turned on (indicated by the symbol "*"). After 70 ticks, this frame is replaced by a blank frame. The end of the item is signaled by the symbol ";".

The expression "%70" is one example of a "switch," as is the clock-on symbol "*". There are over 170 switches that can be embedded in a frame, each of which controls some aspect of the display. For example, the switch "!" means that the current frame does not replace the previous frame but is superimposed on it. Many of these switches are indicated by a single symbol (these guarantee compatibility with scripts written for the older DOS programs); others are enclosed in angle brackets. For example, the switch <line 1> means that the text should be displayed one line below the default display line (the center of the screen). The switch <nfb> specifies that there should be no feedback (the default condition is that, after each response, a message is displayed informing the subject of the correctness of the response and the RT). If the material to be displayed is a graphics file, the switch <gr> is specified, and the material in quotation marks becomes the name of the file to be displayed. Switches in the form of *x*-, *y*-coordinates are provided to position either text or graphics at any location on the screen.

Counters are provided, so that such things as the number of errors or the total number of trials completed can be tracked, and integer arithmetic is provided, enabling the calculation of the mean RT or the current error rate. Conditional branching is possible, so that the item sequence can be controlled by such things as the subject's response to a question or the contents of a counter. Provision is also made for various methods of scrambling the order of items individually for each testing session.

The general aim has been to provide a scripting system that specifies relatively low-level operations, in order to maximize generality. For some unusual applications, this sometimes leads to a fairly cumbersome script, with a special set of switches needing to be included in every item. This is easily coped with by using find and replace commands in Word, although the resulting script may be difficult to read. One example of such an application involves the ability of DMDX to present selected items a second time if they elicited an error the first time. The method involves defining a counter for each item as it is presented and then incrementing it if the response was correct:

```
+100 <set c100 = 0> "HOUSE" *
<IncrementIfCorrect 100>;

−101 <set c101 = 0> "FLIDGE" *
<IncrementIfCorrect 101>;
```

This creates a counter labeled "100" and sets it to zero. The value of this counter is incremented if the current response (i.e., the response to the item "HOUSE") is correct. Subsequently, every item is scheduled for display again but is preceded by an instruction to branch if the appropriate counter has a positive value:

```
<SkipDisplay> 0 <BranchIf 0, c100 .gt. 0>;
+1001 "HOUSE" *;
<SkipDisplay> 0 <BranchIf 0, c101 .gt. 0>;
−1011 "FLIDGE" *;
etc.
```

This first statement causes a branch to the next item numbered "0" if counter 100 is positive (i.e., the next branch statement). Otherwise, control passes to item 1001, where "HOUSE" is presented again with a different item number. The <SkipDisplay> switch at the beginning of each branch statement informs DMDX that this item does not involve any display, and no response is expected from the subject.

Construction of scripts such as this is made easier by using a spreadsheet such as Excel. Using a function such as Concatenate, the above commands can be automatically generated from just the item number and the test item itself. An example of an Excel scripting file is available as part of a package of useful DMDX utilities from the DMDX download site.

In addition, we have compiled a set of example scripts for standard experimental paradigms, such as lexical decision, naming, self-paced reading, picture naming, word identification, masked priming, fMRI scanning, and so forth. This also can be accessed from the DMDX homepage.

An on-line manual for the construction of scripts for the DOS-based DMTG system can be found at the following address: http://www.u.arizona.edu/~kforster/dmastr/dm_man0.htm. Since DMDX is an extension of DMTG, many of the basic features of the DMDX scripts are outlined here. Tutorials on the use of DMDX can also be found on the DMDX homepage.

User assistance. On-line help files for DMDX and TimeDX are included in the download package, and these can also be accessed via the Web at http://psy1.psych.arizona.edu/~jforster/dmdx/help/dmdxhdmdx.htm and http://psy1.psych.arizona.edu~jforster/dmdx/help/timedx htimedxhelp.htm. In addition, there is a DMDX user list serv where users can post queries. Currently, there are about 150 users located in countries such as Australia, U.K., U.S.A., Israel, France, Spain, Germany, China, Taiwan, Japan, and Kazakhstan.

How to Obtain DMDX

The complete DMDX package can be downloaded from the homepage for DMDX, which can be accessed by following the links from the DMASTR Web site: http://www.u.arizona.edu/~kforster/ dmastr/dmastr.htm. This package includes the program TimeDX, which is an essential partner for DMDX and which must be run to select the desired screen resolution and to time the refresh rate. Also included are examples of scripts that execute various tasks.

REFERENCES

Forster, K. I. (1970). Visual perception of rapidly presented word sequences of varying complexity. *Perception & Psychophysics*, **8**, 215-221.

MacInnes, W. J., & Taylor, T. L. (2001). Millisecond timing on PCs and Macs. *Behavior Research Methods, Instruments, & Computers*, **33**, 174-178.

McKinney, C. J., MacCormac, E. R., & Welsh-Bohmer, K. A. (1999). Hardware and software for tachistoscopy: How to make accurate measurements on any PC utilizing the Microsoft Windows operating system. *Behavior Research Methods, Instruments, & Computers*, **31**, 129-136.

Myors, B. (1999). Timing accuracy of PC programs running under DOS and Windows. *Behavior Research Methods, Instruments, & Computers*, **31**, 322-328.

NOTES

1. On some early Windows 95 systems, DirectX was not included. DirectX can be downloaded from the DMASTR site or from Microsoft.

2. Actually, on some machines there are always 998 callbacks per second, but they do not arrive at equal intervals of time.

3. This limit depends on the memory of the video card.

4. Myors (1999) used a similar procedure involving the autorepeat function of the keyboard. This is not possible with DirectX, since autorepeat codes are filtered out.

5. The *SD* that Myors (1999) obtained for the DOS test is nevertheless well below the value of 0.84 reported for DMDX in Table 1. This is perhaps not surprising, since Myors's program was extremely simple and was designed to do nothing other than time the video retrace or the interval between keystrokes (but not both at the same time). In contrast, DMDX was tested while it was carrying out both of these tasks simultaneously and, in addition, accessing the hard drive and assembling the material to be displayed, not to mention monitoring network traffic. It should also be noted that more recent tests running DMDX under Windows ME or XP on newer hardware produced *SD*s that were very close to Myors's DOS values.

(Manuscript received March 16, 2001;
revision accepted for publication August 6, 2002.)