Millisecond-accurate synchronization of visual stimulus displays for cognitive research

SHUIQING XIE, YANG YANG, and ZHONGLE YANG
South-Central University for Nationalities, Wuhan, China

and

JIPING HE
Arizona State University, Tempe, Arizona

A widely adopted approach in cognitive psychology research is to analyze changes in the response time to a stimulus onset in order to infer information about the cognitive functioning of a subject being tested. But current techniques have inherent variations in the timing between stimulus activation and stimulus display of up to tens of milliseconds, thereby introducing significant errors when response time or the latency of neural responses is measured. This article presents a novel yet easy-to-implement solution for improving resolution in the synchronizing of stimulus activation and stimulus display. Unlike traditional methods in which the stimulus onset is set as the time at which the routine for displaying the stimulus is called, this approach uses DirectX to monitor the scan line of CRTs and sets the stimulus onset to the time at which the scan line arrives at the position where the stimulus is to be drawn. This approach removes the uncertainty involved in having a time delay between the activation of the display routine and the actual time at which the display occurs, improving the accuracy of response time and latency period measurements to within 200 μ sec. With a specially developed driver, this solution can generate a trigger signal synchronized precisely with the stimulus onset in all popular Windows systems (including Windows 2000/XP).

Response time and latency period are two often measured parameters in cognitive psychology research. Response time is the interval between the onset of a stimulus and the initiation of a subject's responding action, and latency period measures the interval between the onsets of a stimulus and components of the event-related potentials (ERPs). Precise measurements of the response time and the latency period are critical for distinguishing brain cognition processes accurately and evaluating brain cognition functions objectively.

As research in cognitive psychology becomes ever more sophisticated, the varieties of and display modes for visual stimuli become more complex. The visual stimuli used for these research projects include complex fractal images, besides letters, words, and simple images. The display modes include not only flashing images, but also moving images—for instance, those used in motion perception research (Braddick, Wishart, & Curran, 2002; Raymond, 2000; Xie & Yang, 2002). Even multiple stimuli are sometimes required, and they flicker or move independently at different speeds, as deployed in brain-computer interface researches (Farwell & Donchin, 1988;

Ming, Xiaorong, Shangkai, & Dingfeng, 2002). The sophistication and complexity involved in presenting visual stimuli make the measurement of response time and latency period more and more difficult, and the demand for improved accuracy is increasing. But reaction time and latency period measurements with PCs cannot meet these higher requirements well, because they lack accuracy or require additional hardware.

By definition, both response time and latency are to be measured from the onset of a stimulus. But in practice, the measurement methods implemented with computers typically use the time at which the computer program calls the routine to display the stimuli as the reference time, instead of the time at which the stimuli actually appear. Because display routine activation and stimulus appearance are not simultaneous, there is an appreciable delay between the two events. This delay makes response time and latency period measurements erroneous and variable. There are two compounding factors contributing to the delay. (1) The function used to draw stimuli on a computer monitor may not be executed immediately when it is called, because a computer, which uses a multi-tasking operation system, such as the widely used Windows system, must execute many system service applications besides the user application for experiments. These always interrupt the user application. (2) Even as the function is executed, the stimuli may not appear immediately if the scan line is not at the position on the monitor at which the stimuli will be drawn. This delay is

Correspondence concerning this article should be addressed to Z. Yang, Laboratory of Cognitive Science, South-Central University for Nationalities, Wuhan 430074, China (e-mail: yangzl@scuec.edu.cn).

Note—This article was accepted by the previous editor, Jonathan Vaughan.

caused by the inherent display mode: the top-to-bottom, line-by-line scanning. The scan line may be out of the area where the stimulus will be drawn. It may not have arrived at or may have already left the area. Each of these factors can introduce a delay, and the effects are additive in producing error. The amount of delay is uncertain and can be as short as a few microseconds or as long as tens of milliseconds.

As computer architecture has advanced for both software and hardware, the error due to multitasking has been decreasing. It is now possible to reach a satisfactory level of performance with a negligible time delay in less than 1 msec (André, Ghio, Cavé, & Teston, 2003; Forster & Forster, 2003; MacInnes & Taylor, 2001; McKinney, MacCormac, & Welsh-Bohmer, 1999). However, delay from the second factor will occur as long as current display technology is used. This error varies, and maximum delay depends on the scanning rate of the monitor, which has been discussed in the literature (Bridgeman, 1998; Krantz, 2000). If the monitor frame refresh rate is f , the cycle is $T = 1/f$, which includes the video blanking time Tb . Let's assume that the monitor size and the position of the stimulus display are as shown in Figure 1.

When $y \leq y_0$ (y is above y_0), the time delay induced by the display is $t_{\text{before}} = (T - Tb)(y_0 - y)/h$; when $y \geq y_0 + L$, $t_{\text{after}} = (T - Tb)(h - y + y_0)/h + Tb$. The t_{after} error is obviously larger than that for t_{before} . If stimulus height is 1 pixel and $y = y_0 + 1$, it is easy to show that the time delay occupies the complete cycle of top-to-bottom scanning: $t_{\text{after}} = (h - 1) * T/h + Tb/h \approx T$. Maximum error will be 10 msec if the frame rate is 100 Hz. For cognitive research, 10 msec is a significant delay when response time is measured, since researchers are focused on between-subjects differences of a magnitude of tens of milliseconds. Furthermore, the delay is not fixed among trials, making it impossible to make corrections.

Figure 1. The illustration of a stimulus display on a computer monitor. The height of the display area is h . The display location for the stimulus starts at y_0 , with a vertical dimension of L . The y is the instant location of the scanning line when the display routine executed. The unit is in pixels.

This problem has been virtually resolved under the DOS system (Hamm, 2001; Hamm, Johnson, & Kirk, 2002), but DOS is no longer widely used in research laboratories for experimental control. Developing a simple yet effective method for reducing the uncertainty in time delay and/or the time delay itself directly will improve the ability of scientists and clinicians to investigate cognitive functions with the popular Windows systems.

There have been reports of hardware solutions for decreasing measurement errors. With external hardware, McKinney et al. (1999) have achieved a high timing accuracy of within 100 μsec . However, this approach requires an additional hardware setup that is expensive for many research laboratories. It also requires engineering knowledge and skills for programming and circuit design and, therefore, is not a desirable solution for most cognitive scientists if alternative approaches can be found (McKinney et al., 1999).

We have taken a different approach: using a simple software technique to reduce the uncertainty and time delay by activating the stimulus display after determining where the scan line is on the display monitor. This is a paradigm shift. In contrast to the traditional definition, according to which the stimulus onset is the time at which the display routine is called, we set the stimulus onset at the time at which the scan line arrives at the position where the stimulus is to be displayed (see Figure 1). This approach sets the stimulus onset time so that it is consistent with how the response time is supposed to be measured. We achieved the goal by acquiring real-time position data about the monitor scan line, using DirectX. To achieve a true synchronization of the activation and display of the stimulus, we developed a special driver for accessing computer I/O ports that is applicable in all popular Windows systems (including Windows 2000/XP).

APPROACH

Acquisition of Scan Line Position and High-Precision Time

DirectDraw is a key component in the DirectX application programming interface (API) that allows direct manipulation of display memory, the hardware blotter, hardware overlay support, and flipping surface support. DirectDraw is a software interface that provides direct access to display devices while maintaining compatibility with the Windows graphics device interface (GDI) and existing Microsoft Windows-based applications and device drivers. The API function for returning the position of the scan line is `IDirectDraw::GetScanLine`. The syntax is

```
HRESULT GetScanLine (LPDWORD lpdwScanLine).
```

Information about the scan line position is reserved in `lpdwScanLine` (Microsoft DirectX Documentation for C++, 2002).

It is important to note that the ordinary API time functions in the Windows system do not provide the time res-

olution required for cognitive experiments. For example, `GetTickCount()` can provide a time resolution of only 1 msec at best, since it is limited by the resolution of the system timer (Microsoft MSDN Library for Visual Studio .NET 2003, 2003). There is a high-resolution timer, a common chip on most mother boards, which operates autonomously and independently of whatever application is currently running. If a high-resolution performance counter exists on the system, the `QueryPerformanceFrequency` function can be used to express the frequency, in counts per second. The value of the count is processor dependent. On some processors, for example, the count might be the cycle rate of the processor clock (Microsoft MSDN Library for Visual Studio .NET 2003, 2003). Therefore, the resolution is more than adequate for experimental needs. The `QueryPerformanceCounter` function retrieves the current value of the high-resolution performance counter. If the application calls `QueryPerformanceCounter` immediately before and immediately after a section of code, the time elapsed to execute the code can be calculated from the difference of the two count values returned from `QueryPerformanceCounter` and the frequency value returned from `QueryPerformanceFrequency`. The two functions' syntaxes are

```

    BOOL QueryPerformanceCounter (
        LARGE_INTEGER *lpPerformancecount
        // counter value
    );
and
    BOOL QueryPerformanceFrequency (
        LARGE_INTEGER *lpPerformanceFrequency
        // frequency value
    ).

```

A Special Driver for I/O Port Access

In modern psychology experiments, two computer systems are frequently used, one for stimulus display and the other for measurements of latency period and response time. The two computers have to be synchronized. The computer recording responses for the measurement of latency period and response time must receive a triggering signal when the stimulus is drawn on the screen of the other computer controlling behavior events. This triggering is usually sent through the parallel port. It has been easy to execute this operation under Windows 98 by using a simple routine written in assembly codes. This approach cannot work under Windows 2000/XP, which denies the application direct access to an I/O port. We have developed a special driver for overcoming this problem. This driver is a SYS file that makes any I/O port directly accessible for applications. This driver can be loaded dynamically, without the need to restart the computer, because it does not require any new hardware and will not edit the Windows register, a feature desirable for easy use and convenience in designing and performing experiments.

When the special driver is used to generate a triggering signal and to send over a selected I/O port, determined by the experimenter's choice or the hardware setup, the application program needs to complete four procedure calls: driver load and initialization, port reading, port writing, and driver unload and cleanup. We wrote four functions to carry out these tasks in a strict order sequence:

```

    BOOL OpenMyPort(),
    for loading and initializing the driver;
    BYTE ReadPortByte(WORD port),
    for reading the status of the port;
    void WritePortByte(WORD port, BYTE data),
    for writing to the port;
and
    BOOL CloseMyPort(),
    for clearing and closing the port before the
    application program is terminated.

```

When the `ReadPortByte` or `WritePortByte` functions are used to read from or write to an I/O port, a key step is to call the `DeviceIoControl` function (Microsoft Windows DDK Documentation, 2001). This function's syntax is

```

    BOOL DeviceIoControl(
        HANDLE hDevice,
        // Handle to the device on which the operation
        // is to be performed.
        DWORD dwIoControlCode,
        // Control code for the operation.
        LPVOID lpInBuffer,
        // Pointer to the input buffer that contains the
        // data required to perform the operation.
        DWORD nInBufferSize,
        // Size of the input buffer, in bytes.
        LPVOID lpOutBuffer,
        // Pointer to the output buffer that is to receive
        // the data returned //by the operation. The format
        // of this data depends on the value of the dwIoCon-
        // trolCode parameter.
        DWORD nOutBufferSize,
        // Size of the output buffer, in bytes.
        LPDWORD lpBytesReturned,
        // Pointer to a variable that receives the size of
        // the data stored in the output buffer, in bytes.
        LPOVERLAPPED lpOverlapped
        // Pointer to an OVERLAPPED structure
    ).

```

In the driver developed above, the operations that read from and write to the I/O port are carried out by two specially defined I/O control codes. The two I/O control codes are `IOCTL_MYPORT_READ_BYTE` and `IOCTL_MYPORT_WRITE_BYTE`. When `ReadPortByte` calls

Figure 2. Illustration of the testing method.

DeviceIoControl, the dwIoControlCode parameter is IOCTL_MYPORT_READ_BYTE, and when WritePortByte calls DeviceIoControl, the dwIoControlCode parameter is IOCTL_MYPORT_WRITE_BYTE.

Finally, it is important to call the CloseMyPort function to stop service and unload the driver before the application is terminated. The driver and the program codes of the four functions are available upon request.

Test and Validation of the Approach

Program codes are written using C++ and are compiled under Visual Studio 6.0. The operating system tested includes Windows 98 and Windows 2000/XP. The hardware configuration of the computer is Pentium III 733 MHz with SDRAM 256 MB and the video card Matrox Millennium G400 with a 32-MB display memory. The measurement tool is a double-channel oscilloscope (Tektronix Model TDS210).

In order to check whether the generation of triggering and the display of the stimulus are truly synchronized, we designed a special setup (shown in Figure 2). The triggering signal and the video signal, relative to the stimulus, were sent simultaneously to the two separate channels of a high-performance memory oscilloscope; then we examined the two signals' rising edge to measure the time delay. The stimulus was a red line with a dimension of 200×1 pixels, and the relative video signal was recorded from red component video signal wire (Pin 1 in Figure 3A); the triggering, relative to the stimulus, was a pulse with a width of 10 msec, recorded from parallel port data bit 0 (Pin 2 in Figure 3B).

RESULTS AND DISCUSSION

The results are shown in Figure 4: The signal from Oscilloscope Channel 1 is the triggering signal, and the signal from Channel 2 is the red component video signal.

Under Windows 98, we used a monitor with a display resolution of $1,024 \times 768$ pixels and a color depth of 32 bits per pixel. When the frame rate was set at 60 Hz, the interval of the two rising edges on the oscilloscope

varied between 180 and 200 μsec . When the frame rate was set at 75 Hz, the interval varied between 170 and 200 μsec . For the frame rate of 85 Hz, the interval varied from 175 to 195 μsec .

For the same monitor under Windows 2000, the results were very similar, with a slight improvement. For the frame rate of 60 Hz, the interval of the two rising edges on the oscilloscope varied from 175 to 200 μsec ; for the frame rate of 75 Hz, the interval varied from 150 to 180 μsec ; and for the frame rate of 85 Hz, the interval varied from 155 to 185 μsec .

Similar results were also obtained under Windows XP. For the frame rate of 60 Hz, the interval varied from 175 to 195 μsec ; for the frame rate of 75 Hz, the interval var-

Figure 3. (A) 15-pin VGA port. (B) 25-pin parallel port.

Figure 4. Illustration for testing results. Channel 1 is the triggering signal. Channel 2 is the response signal from the red component of the video card for displaying the stimulus. It has a range of variation of about 30 msec after a delay of 135 msec (t_1) from the trigger.

ied from 170 to 190 μsec ; and for the frame rate of 85 Hz, the interval varied from 135 to 165 μsec .

From these results, we can conclude that the interval between the trigger and the red video signal fluctuates slightly within a small range of 20–30 μsec , independently of the monitor frame rate or OS. The maximum error is only 200 μsec , a significant improvement over the traditional approach by an order of two. It is also much simpler or more economical than a hardware implementation with a comparable performance. This inaccuracy can be further improved with a simple modification—for example, the inaccuracy of 180–200 μsec will be $-10 \mu\text{sec} \sim 10 \mu\text{sec}$ by subtracting 190 μsec .

CONCLUSION

Our analysis indicates that an important source of inaccuracy in measurement of response time and latency period is the uncertainty of the scan line position in the top-to-bottom scanning of a monitor display mechanism when a stimulus display is activated. The inaccuracy will be up to 10 msec even when the monitor refresh rate is as high as 100 Hz. This magnitude of inaccuracy, especially the uncertainty and the range of variation up to 10 msec, cannot be ignored in cognitive research, where results often depend on individual differences in response time. This accuracy and uncertainty can be reduced to a negligible level under DOS, but this solution will lose its practicality with DOS's exit in the updated Windows systems (Windows 2000/XP). The solution based on hardware development has generated satisfactory improvement in timing accuracy; however, it requires a substantial investment in new equipment, a cost prohibitive to the general research community. We have developed a cost-effective approach to improving accuracy in measuring response time and latency period with a resolution close to what has been achieved by the hardware solution.

In this software solution, to decrease inaccuracy, we shifted the paradigm for measuring response time by monitoring the exact position of the display cursor in real time and setting the stimulus onset time when the cursor arrives at the location where stimulus display is to appear. We use DirectX to get the position of the display cursor in real time. This allows us to take the appearance of the stimulus as the timing onset and to generate a synchronous triggering. Using a special port driver, the operation of generating the triggering can be carried out under most Windows OSs (including Windows 2000/XP). When this software solution is used, the interval between the triggering (stimulus onset) and the appearance of the stimulus is 135–200 μsec under different Windows OSs with different monitor frame rates, a result with a much reduced uncertainty range ($<60 \mu\text{sec}$) and also a maximum error of no more than 200 μsec . With a correction algorithm, the absolute error can be further reduced to less than 30 μsec . The most significant advantage of this approach is that it does not require any additional equipment, while achieving an excellent level of performance.

Availability

All the data and pictures in this article can be used by anyone if he/she informs the author. Anyone can get the driver and part of the soft code by e-mailing the author (yangzl@scuec.edu.cn or xsq_hao@163.com).

REFERENCES

- ANDRÉ, C., GHIO, A., CAVÉ, C., & TESTON, B. (2003). PERCEVAL: A computer-driven system for experimentation on auditory and visual perception. In *Proceedings of the XVth International Congress of Phonetic Sciences* (Barcelona), pp. 1421–1424.
- BRADDICK, O. J., WISHART, K. A., & CURRAN, W. (2002). Directional performance in motion transparency. *Vision Research*, **42**, 1237–1248.
- BRIDGEMAN, B. (1998). Durations of stimuli displayed on video display terminals: $(n - 1)/f + \text{persistence}$. *Psychological Science*, **9**, 232–233.

- FARWELL, L. A., & DONCHIN, E. (1988). Talking off the top of your head: Toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography & Clinical Neurophysiology*, **70**, 510-523.
- FORSTER, K. I., & FORSTER, J. C. (2003). DMDX: A Windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers*, **35**, 116-124.
- HAMM, J. P. (2001). Object-oriented millisecond timers for the PC. *Behavior Research Methods, Instruments, & Computers*, **33**, 532-539.
- HAMM, J. P., JOHNSON, B. W., & KIRK, I. J. (2002). Comparison of the N300 and N400 ERPs to picture stimuli in congruent and incongruent contexts. *Clinical Neurophysiology*, **113**, 1339-1350.
- KRANTZ, J. H. (2000). Tell me, what did you see? The stimulus on computers. *Behavior Research Methods, Instruments, & Computers*, **32**, 221-229.
- 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 tachistoscopes: How to make accurate measurements on any PC utilizing the Microsoft Windows operating system. *Behavior Research Methods, Instruments, & Computers*, **31**, 129-136.
- MICROSOFT DIRECTX DOCUMENTATION FOR C++, DIRECTX SDK (2002).
- MICROSOFT MSDN LIBRARY FOR VISUAL STUDIO .NET 2003 (2003).
- MICROSOFT WINDOWS DDK DOCUMENTATION, WINDOWS DDK 2600 (2001).
- MING, C., XIAORONG, G., SHANGKAI, G., & DINGFENG, X. (2002). Design and implementation of a brain-computer interface with high transfer rates. *IEEE Transactions on Biomedical Engineering*, **49**, 1181-1186.
- RAYMOND, J. E. (2000). Attentional modulation of visual motion perception. *Trends in Cognitive Sciences*, **4**, 42-50.
- XIE, Y., & YANG, Z. L. (2002). Event-related potentials during imitated natural reading. *International Journal of Psychophysiology*, **45**, 422-424.

(Manuscript received December 12, 2003;
revision accepted for publication August 9, 2004.)