# LATENCY MEASUREMENT OF A REAL-TIME VIRTUAL ACOUSTIC ENVIRONMENT RENDERING SYSTEM

*Joel D. Miller*[1] *Mark R. Anderson*[1] *Elizabeth M. Wenzel*[2] *Bryan U. McClain*[3]

Spatial Auditory Displays Lab
NASA Ames Research Center
Mail Stop 262-6
Moffett Field, CA 94035-1000 USA
jdmiller@mail.arc.nasa.gov
[1]QSS Group Inc., [2]NASA Ames Research Center, [3]San Jose State University

**ABSTRACT**

Techniques for measuring and estimating the end-to-end latency and component latencies of a virtual acoustic environment are discussed. These key parameters impact the responsiveness and, hence, "realism" of a virtual environment.

## 1. INTRODUCTION

Latency provides an important indicator of the dynamic performance of a virtual acoustic environment (VAE) and it is critical that it be carefully defined and measured. In a VAE, the end-to-end latency refers to the time elapsed from the transduction of an event or action, such as movement of the head, until the consequences of that action cause the equivalent change in the virtual environment. Latencies are contributed by individual components of the system, including tracking devices, signal processing algorithms, device drivers, and communication lines. Due to variability in the way these components interact, a system's end-to-end latency will vary over time. Thus, measurements of the mean, standard deviation, and range are needed to characterize this parameter.

Psychoacoustic data can provide guidelines regarding whether a given system's end-to-end latency meets perceptual requirements [1]. For example, examination of the head motions that listeners use to aid localization suggests that the angular velocity of some head motions (in particular, left-right yaw) may be as fast as 175°/s for short time periods (about 1s). From psychophysical studies of the minimum audible movement angle for real sound sources (listener position fixed), one can infer that the minimum perceptible end-to-end latency for a virtual audio system should be no more than about 70ms for a source velocity of 180°/s. If one assumes that these thresholds are similar for all kinds of relative source-listener motion (e.g., when the source is fixed and the listener is moving), then latencies greater than 70ms may exceed the perceptible threshold during active localization. Such latencies could potentially result in short-term under-sampling (compression) of relative listener-source motion as well as positional instability of the simulated source.

## 2. LATENCY MEASUREMENT TOOLS

In this study, the latency of the VAE rendering system SLAB [2] is analyzed. Since SLAB is developed under Windows 2000 (aka Win2k), some of the following latency measurement techniques use Microsoft Windows APIs (application programming interfaces). The features used, however, are fairly standard and should be available on other platforms. Source code demonstrating these techniques is available to the public as part of the SLAB User Release [2].

When measuring latency, one needs an accurate way to measure the interval of time between two events. These events can be inside or outside of a computer. VAE end-to-end latency is an example of an interval between two external events where the first event is the user crossing a threshold and the second event is the user hearing the rendered result of the threshold crossing. API latency is an example of an interval between an internal event and an external event where the first event is the time at which an API call is made and the second event is the user hearing the rendered result of the API call.

### 2.1. External Events

To measure the interval between events outside of a computer, an interval counter or digital storage oscilloscope can be used to measure the time difference between rising edges of two electrical signals. In some cases, a transducer is required to convert the event of interest into an electrical signal (e.g., an optical switch to capture the time at which an object crosses a physical threshold).

### 2.2. Internal Events

When measuring event intervals inside of a computer, time functions can be used to time stamp events. The difference between event time stamps then provides the interval value. Often, multiple time functions exist, so one must be careful to select the timer with the greatest accuracy and resolution.

In the Win32 SDK (software development kit), the `QueryPerformanceCounter()` function provides an extremely accurate timer with resolution of a microsecond or better (depends on OS and CPU).

### 2.3. Mixed External and Internal Events

To measure the interval between an event inside and an event outside of a computer, the internal event needs to be externalized. An internal event can be externalized by writing to the serial port or the parallel port when the internal event occurs. Of course, the latency of the port write must be determined and stable.

#### *2.3.1. Serial Port Externalization*

For this study, the serial port was the preferred technique for externalizing an internal event because it is supported under both Win98/ME and Win2k. The drawback compared to the parallel port is increased difficulty in use and increased latency. Once the serial port is configured with `CreateFile()`, a rising edge can be created on pin 3 of a 9 pin serial port with the call:

```
char  chr = 0x00;
WriteFile( hCom, &chr, 1, &numWrite, NULL );
```

Serial port loopback tests under Win98, WinME, and Win2k on systems ranging from 450MHz to 1.5GHz yielded approximately a 0.5ms write/read time. Thus, the internal event is externalized in less than 0.5ms.

#### *2.3.2. Parallel Port Externalization*

Although the parallel port is extremely easy to use and has very low latency under Win98/ME, it is difficult to use under Win2k. Under Win98/ME, a rising edge can be created on pin2 of the parallel port with the two calls:

```
_outp( 0x378, 0 );
_outp( 0x378, 1 );
```

Under Win2k, the `_outp()` instruction is a privileged kernel mode instruction. Since the user's code executes outside of the kernel, the `_outp()` instruction will cause an exception error when executed under Win2k.

For those using Win98/ME or another operating system where the `_outp()` call is allowed (e.g., Linux with root privilege), the parallel port may very well be the preferred option. Parallel port loopback tests under WinME on a 450MHz machine yielded a 13µs write/read time. Thus, the internal event is externalized in less than 13µs.

## 3. SLAB LATENCY COMPONENTS

To measure SLAB's latency, two approaches were taken, a low-level individual latency component analysis, and a high-level user parameter analysis. In the low-level approach, each contributing component was isolated and analyzed. In the high-level approach, end-to-end latency data was collected for several permutations of SLAB user parameters. In this section, the low-level perspective will be discussed. The high-level perspective will be discussed in the following section.

### 3.1. Latency Measurement

A swing-arm apparatus [3] was used to measure tracker latency and end-to-end latency. An electromagnetic Polhemus Fastrak tracker sensor is attached to a mechanical swing-arm. When the swing-arm is pushed, it passes through an optical switch (Figure 1, Ch1), triggering a single-shot oscilloscope capture of tracker serial output (Figure 1, Ch2), tracker library output (Figure 1, Ch3), and SLAB headphone output (Figure 1, Ch4).

The SLABLatency utility monitors the location of the sensor using an in-house tracker driver. SLABLatency is configured ahead of time to know the location of the optical switch. Thus, at the same moment the optical switch is activated, SLABLatency changes the SLAB scenario, causing a change in the headphone display. The interval from optical switch pulse to headphone display change is SLAB's end-to-end latency (Figure 1, "C1->C4 Dly").

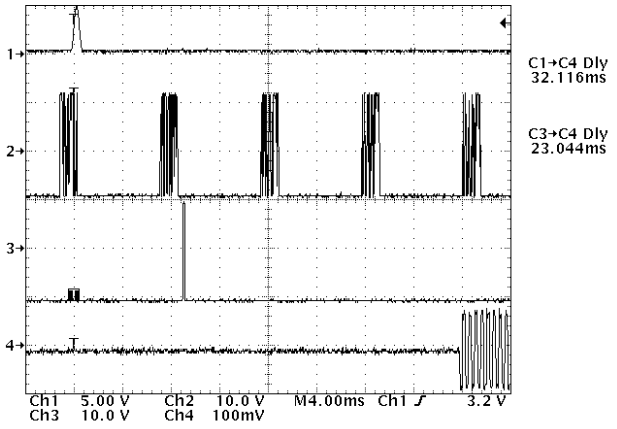


Figure 1. *Latency measurement oscilloscope screenshot. Ch1: optical switch, Ch2: tracker serial communications, Ch3: tracker library serial port write, Ch4: SLAB headphone display. Output buffer size = 4096 bytes, write buffer size = 256 bytes.*

### 3.2. Tracker Latency

Two components contribute to Fastrak tracker latency, tracker update rate and EM (electromagnetic) field sampling.

#### *3.2.1. Tracker Update Rate Latency*

The tracker was configured to operate at its highest update rate, 120Hz (update period = 8.3ms). Communication with the tracker occurred over an RS232 serial connection. In streaming mode, the tracker generates steady bursts of serial data at 8.3ms intervals (Figure 1, Ch2). Since the optical switch can be crossed at any time within this interval, a variable latency exists of **0.0-8.3ms** with a uniform probability of any given latency value occurring at any given time.

#### *3.2.2. Tracker Electromagnetic Field Sampling Latency*

Attaching an electromagnetic pickup to the Fastrak source revealed that the tracker locates the sensor by sampling three EM bursts from the source. The three bursts last 1.5ms each. Since the sensor's location can vary slightly between the three bursts, the midpoint of the middle burst is defined as the EM field sampling time. The EM sampling latency is the interval from this point to the beginning of serial output. This interval was measured to be **3.5ms**.

### 3.3. Serial Communication Latency

The tracker is configured to communicate with a computer over a 115,200 bps (bits per second) RS232 serial line. The sensor location data packet consists of 17 data bytes (8 bits per byte). Each serial data byte contains an additional start bit and stop bit. Thus, the total number of bits transmitted per data packet is 170 bits. At 115,200 bps, the serial communication latency is: 170 bits / 115,200 bps = **1.5ms**. This is consistent with the measured width of the serial bursts in Figure 1, Ch2.

### 3.4. Tracker Driver Latency

The tracker driver latency was measured using serial port externalization. Inside the SLAB API update loop, the tracker driver blocks, waiting for a tracker data packet. When new data is available, the driver unblocks and calls a SLAB API function with the new location data. A serial port write was inserted between the blocked tracker driver function and the SLAB API function. The tracker driver latency is the difference between the last serial bit read (Figure 1, Ch2) and the serial port write (Figure 1, Ch3). Although it is difficult to see in Figure 1, using the appropriate time base, the tracker driver latency was measured to be **0.4-0.5ms**.

### 3.5. API Latency

The API latency is the interval between an API scenario update and the display of the rendered result of that update (Figure 1, "C3->C4 Dly"). In SLAB, this is largely a function of the processing frame size, the sound output buffer management algorithm, and the sound peripheral driver latency. A fourth contributing factor can be the processing parameter smoothing technique, but for the purposes of this analysis, parameter smoothing is not considered (parameter smoothing is disabled using the SLAB API call `SmoothTime(0.0)`).

### 3.6. Frame Size Latency

Frame size (aka block size) refers to the number of sound samples processed at a time. When a new frame of input is available, scenario parameters (e.g., listener position) are converted to processing parameters (e.g., filter taps). Excluding parameter smoothing, processing parameters do not change during the processing of the frame. Thus, the entire frame is processed using one set of scenario parameters. If a scenario update occurs just prior to computing the frame, little latency is introduced. However, if the scenario update occurs just after a frame is processed, a frame size amount of latency is introduced. As frame size increases, scenario update rate decreases and API latency and API latency jitter increase. Unfortunately, small frame sizes are costly to compute, so a trade-off exists between CPU usage and latency.

In SLAB, the frame size is 32 samples (sampling rate = 44,100 samples/s). To preserve CPU resources, the head-related impulse responses (HRIRs) are updated every other frame, yielding a maximum frame latency of 1.5ms. Since the timing of the scenario update is unconstrained, the frame size latency is stochastic and uniformly distributed between 0.0-1.5ms. As will be seen in the next section, this latency is absorbed within the sound output buffer management algorithm.

### 3.7. Output Buffer Latency

The Microsoft DirectSound API provides a low-latency interface to a sound output peripheral. After a frame of samples is processed, it is transferred to a DirectSound output buffer. Selecting the size of the output buffer depends on system load. If the CPU is heavily taxed, the Windows scheduler may starve SLAB, resulting in a buffer underflow and, possibly, an audible artifact. Large buffer sizes help protect against this situation because samples continue to play while SLAB awaits attention from the scheduler. An output buffer size (OBS) between 4096 bytes (23.2ms) and 8192 bytes (46.4ms) is usually sufficient to protect against underflow.

An additional buffer, the write buffer, exists to optimize CPU use. Since DirectSound management can be computationally expensive, the write buffer collects multiple frames of data before copying samples to DirectSound. The write buffer size (WBS) is typically 256 bytes (2 frames, 1.5ms) or 512 bytes (4 frames, 2.9ms).

To analyze the expected latency impact of the output buffer management algorithm two assumptions will be made:

(1) Since the output buffer exists to absorb unexpected CPU usage spikes, most of the time it should be full or near full. It will be assumed it is always as full as the buffer management algorithm will allow.

(2) In theory, a scenario update can occur between any two frames. It will be assumed that rendering the SLAB scenario consumes negligible CPU resources (e.g., one sound source in an anechoic simulation). Thus, the frame processing thread will process all the frames it can until it is forced to stop due to a full output buffer. The result of this assumption is that all frames in the write buffer are processed with the same scenario parameters.

Given assumption 1, after the write buffer is copied to the output buffer, the output buffer is full. Thus, processing suspends awaiting space in the output buffer. Given assumption 2, the write buffer basically becomes the frame, with the write buffer size replacing the frame size. Hence, a latency of 0.0-WBS (ms) is introduced after an API scenario update. Once a write buffer amount of samples have played out of the output buffer, all frames that fit within a write buffer are processed and copied to the output buffer and the suspend-and-fill cycle repeats. This behavior results in a constant latency of (OBS - WBS) existing in addition to the write buffer latency. Since the timing of the scenario update is unconstrained, the resultant output buffer latency is stochastic and uniformly distributed between **(OBS - WBS) and OBS**. The mean of this range will be termed the Estimated Buffer Latency (EBL). As will be seen in the next section, EBL can serve as a rough approximation of API latency. The jitter introduced by the output buffer management algorithm is equal to the write buffer size.

### 3.8. Sound Peripheral Driver Latency

DirectSound can use one of two driver models, VxD (Win98/ME) or WDM (Win98/ME/2k). The WDM driver may include a component called the KMixer that can add up to 30ms of latency. Subtracting the Estimated Buffer Latency from measured API latency yielded **2.1ms** for a driver not using the KMixer (Creative Audigy, Win2k) and 26.3ms for a driver using the KMixer (M Audio Delta66, Win2k).

### 3.9. Client/Server Communication Latency

Thus far, it has been assumed the API latency is confined to the host computer (i.e., the API call occurs on the same computer rendering the virtual environment). In client/server mode, the API call occurs on a client computer while rendering takes place on a separate server computer. In SLAB, a TCP/IP sockets interface is used to communicate between the client and the server over a dedicated 100MBit Ethernet connection.

The client/server communication latency was measured by time stamping the socket packet sends and receives. The client and server timers were synchronized using serial port

externalization. The measured latency values clustered about four points, **0.1ms** (33%), 0.5ms (17%), 1.2ms (33%), and **1.5ms** (16%), with outliers as high as 3ms.

The client and server software can be executed on the same computer in a "pseudo" client/server mode. In this mode, the measured latency was 0.2ms.

### 3.10. Estimated End-to-End Latency

Summing the component latencies from the preceding sections (values in **bold type**) yields the following estimates for host-mode end-to-end latency (in ms, excluding the KMixer):

$t_{min}$ = 0.0 + 3.5 + 1.5 + 0.4 + OBS - WBS + 2.1
$t_{max}$ = tracker update period + 3.5 + 1.5 + 0.5 + OBS + 2.1
$t_{avg}$ = ($t_{min}$ + $t_{max}$) / 2
= 7.6 + tracker update period/2 + OBS - WBS/2

The latency values should be distributed in a trapezoidal or triangular distribution due to the cascade of the uniformly distributed tracker update rate and output buffer latencies. This distribution provides an estimate of the latency jitter in the system. For client/server mode, the following offsets should be added to the estimates above: 0.1ms to $t_{min}$, 1.5ms to $t_{max}$, and 0.8ms to $t_{avg}$.

## 4. END-TO-END LATENCY MEASUREMENTS

End-to-end latencies were measured using the swing-arm apparatus described in Section 3.1 for a VAE system consisting of a Dell Workstation PWS340 (Pentium 4, 2.2 GHz, 512 MB RAM, Windows 2000 SP2), a Creative Audigy sound card (driver e10kx2k.sys v5.12.01.0129-1.00.0010), and a Polhemus Fastrak head-tracker (update rate = 120 Hz). SLAB v5.0.1 was used to convolve an internally generated input signal with a 128-pt. HRIR in a single sound source anechoic simulation.

End-to-end latencies were measured for the three system modes described in section 3.9 (host mode, client/server mode, and pseudo client/server mode). For each mode, 25 latency measurements were taken for different combinations of the SLAB API parameters: output buffer size (4096 and 8192 bytes) and write buffer size (128, 512, 1024 and 2048 bytes; see section 3.7). Measurements were also made for output buffers of 1024 and 2048 bytes. However, it was observed that these smaller buffer sizes resulted in significant buffer underflows and audible artifacts. Thus, data are only reported for the larger output buffers that would be usable in practice.

Since the measured latencies for the three system modes were within 0.8ms of one another, only the host-mode data is displayed in Figure 2. To aid in the interpretation of the results, the values in Figure 2 are plotted as a function of Estimated Buffer Latency (EBL). As described in section 3.7, EBL represents the mean latency due to the output buffer size (OBS) and the write buffer size (WBS):

EBL = (OBS (bytes) - WBS (bytes) / 2) /
(44.1 samples/ms * 2 (bytes/sample)/ch * 2 ch)
= (OBS (bytes) - WBS (bytes) / 2) / 176.4 bytes/ms

The cluster of points on the left side of Figure 2 refers to an OBS of 4096 while the cluster on the right refers to an OBS of 8192. For data points within these clusters, the WBS values from left-to-right are: 2048, 1024, 512 and 128.

In general, the data show that the predicted and empirical values match within less than 1 ms. Further, end-to-end latencies are about the same no matter which system mode is used, as long as a dedicated network is available for client/server mode. For a given tracker update rate, the pattern of the data also indicates that the OBS largely impacts the mean latency, while the WBS affects the latency jitter (i.e., the range of possible latency values). Larger WBS values produce more jitter but a smaller mean latency. Thus, there is a trade-off between jitter and latency for decreasing values of the WBS. It should be remembered that the tracker update rate has a significant impact on both mean latency and jitter. For example, if the tracker update rate is reduced, mean latency would increase by a fixed amount (i.e., all data points would shift upward) and the overall latency jitter would also increase.

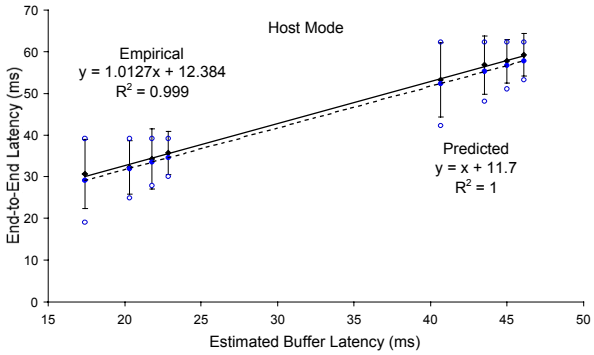


Figure 2. *Predicted and measured end-to-end latency. Circles: predicted minimum and maximum latency ($t_{min}$, $t_{max}$). Points: predicted average latency ($t_{avg}$). Diamonds: mean measured latency. Error bars: ±2 standard deviations for the empirical data. Linear best fits for the mean predicted (dashed line) and empirical latencies (solid line) are indicated.*

Thus, in order to minimize both latency and jitter, the data suggest that a user should choose the fastest tracker update rate possible and the smallest OBS and WBS values that avoid buffer underflow.

## 5. CONCLUSIONS

This paper describes a set of tools for performing high-precision latency measurement. Using these tools, formulas were derived to characterize and predict the end-to-end latency of SLAB, a real-time VAE rendering system. The accuracy of the formulae was verified by comparison to empirical data. Future work will include analyzing the effects of system load, parameter smoothing, and alternate buffer management techniques on latency and latency jitter.

## 6. REFERENCES

[1] E.M. Wenzel, "The role of system latency in multi-sensory virtual displays for space applications," *Proc. HCI Intl.*, New Orleans, LA, August 2001, pp. 619-623.
[2] http://human-factors.arc.nasa.gov/SLAB
[3] B.D. Adelstein, E.R. Johnston, and S.R. Ellis, "Dynamic Response of Electromagnetic Spatial Displacement Trackers," *Presence*, vol. 5, no. 3, pp. 302-318, 1996.

**Acknowledgements:** Funding for this work was provided by the Airspace Operations Systems (AOS) Project of NASA's Airspace Systems Program.