Improved temporal response in virtual environments through system hardware and software reorganization

Richard H. Jacoby, Bernard D. Adelstein*, and Stephen R. Ellis**

Sterling Software/NASA Ames Research Center
*University of California, Berkeley/NASA Ames Research Center
**NASA Ames Research Center
Moffett Field, CA

ABSTRACT

Excessive end-to-end latency and insufficient update rate continue to be major limitations of virtual environment (VE) system performance. Beginning from a typical baseline VE in which a spatial tracker is polled to deliver data via an RS-232 interface at each update of a single application program, we examined a series of hardware and software reconfigurations with the aim of reducing end-to-end latency and increasing update rate. These reconfigurations included: 1) multiple asynchronous UNIX processes communicating via shared memory; 2) continuous streaming rather than polled tracker operation; 3) multiple rather than single tracker instruments; and 4) higher bandwidth IEEE-488 parallel communication between tracker and computer. Starting from an average latency of 65 msec and an update rate of 20 Hz for a standard 1000 polygon test VE, our most successful implementation to date runs at 60 Hz (the maximum achievable with our graphics display hardware) with approximately 30 msec average latency. Because our equipment and architecture is based on widely available hardware (i.e., SGI computer, Polhemus Fastrak) and software (i.e., Sense8 WorldToolKit), our techniques and results are broadly applicable and easily transferable to other VE systems.

Keywords: Virtual Environments, Virtual Reality, Latency, Delay, Update Rate, Real Time, Trackers

1. INTRODUCTION

Inadequate dynamic response due to excessive latency and insufficient update rates remains a stumbling block to the implementation of effective virtual environment (VE) applications. While enhancements to computing capability have offered steady significant increases in specified polygon fill rates, improvements in model computation and drawing speed affect only a portion of the factors that determine overall VE update rates and latencies. Thus, these computer hardware improvements can have only limited impact on the VE dynamic response characteristics experienced by the user.

One approach to ameliorating poor VE dynamic response has been the introduction of prediction algorithms to compensate for the latency (i.e., delay) between input action and rendered output. While reductions in apparent time delay have been reported for a variety of prediction schemes [1, 2, 3, 4, 5], predictors increase noise and generate overshoot not originally present in the sensed input to the VE, and thus may degrade human operator performance. Furthermore, prediction does not benefit VE update rates—in fact, the added computational burden of a poorly implemented predictor can result in slower rates.

In general, a VE system is composed of the hardware and software of input transducers (e.g., motion trackers, gloves, keyboards, joysticks, switches, voice actuators, etc.), one or more computer processors which use this input information to modulate the behavior and content of the VE, and output rendering devices which display information to one or more (e.g., visual, aural, haptic) human senses. Because each of these components either operates on or produces sampled data, each has its own inherent operating cycle which consumes a finite amount of time and thereby contributes to latency. Of equal significance when considering VE dynamic response is the timing of the communication paths between the various input, processing, and output components.

In the work presented here, we have been able both to reduce latency *and* increase update rates as a consequence of examining the detailed timing characteristics of individual VE components and intercomponent communication. Significant improvements in dynamic response were achieved through a reorganization of system software and hardware that eliminates a number of "inefficiencies" typical to the configuration of most VE systems. The resulting performance enhancements do not entail heightened noise or overshoot. Though one cannot attain zero latency through the reorganization measures we describe be-

Proceedings of SPIE, 2653: *Stereoscopic Displays and Virtual Reality Systems III*, 1996, pp.271-284.

low, the results of this work—shortening the actual time delay to be compensated and making data available more frequently through quicker update rates—are essential to improved predictor implementations.

Though we had the general structure for our final VE system configuration in mind at the outset, the software and hardware reconfiguration described in this paper was pursued as a sequence of steps. At each step, modifications to VE software or hardware components were followed by rigorous measurement of the resultant changes in system performance. Thus, the work was, in a sense, iterative with the measurement analysis at each step suggesting the next system modifications to be considered.

The remainder of this paper is organized as follows. First, we describe apparatus and procedures for measurement of end-to-end latency and effective update rates in a VE system. Next, beginning from a baseline configuration, we present the major software and hardware modifications at each step of the reorganization. At each step we consider the implications of the measured performance for subsequent system reorganization. A summary of the results and observations from all configurations concludes the paper.

2. DYNAMIC RESPONSE MEASUREMENT

2.1 Definitions

Two fundamental characteristics need to be known to quantify temporal dynamic response of the particular VE input-output (transducer-processor-display) pathway of interest. These are: 1) end-to-end latency and 2) effective update rate.

End-to-end latency for a given VE system pathway is the time elapsed from transduction of an event or action by the input device until the consequences of that action are first made available in the display. It is the sum of the series of latencies, or delay between input and output, for the individual elements comprising the data pathway. As noted earlier these elements include hardware components and their associated software as well as intercomponent communication. We further delineate *internal* latency to include only those elements housed physically inside the computer. In the case of the VE system described in this paper, internal latency is the time between completed receipt of tracker data at the computer's input port and the instant the data begins to be issued at the video port for scanout on the display CRT.

Because individual hardware and software components can each update according to their own cycle timing and since there may be no synchronization between particular VE system components, the effective update rate could be considered nominally as the rate of the slowest component in the pathway of interest. We also recognize that the update rates associated with an individual component, or for the complete VE system, may vary with time. For example the computer polling of an external sensor may occur at a nonuniform rate. Furthermore, under the scheduling structure of the UNIX environment, software cycle times can be highly nondeterministic. Thus, we consider the *average* effective update rate to be a suitable measure of performance.

2.2 Update rate measurement

Update rate measurement of software internal to the computer is a trivial procedure requiring a time stamp function to be called at the same point in the source code for each software cycle update.[1] The difference between time stamp reports for consecutive software cycles is the instantaneous cycle time. Averaging sampled cycle times over many cycles is necessary to account for the underlying stochastic variation of the process being observed, especially in a UNIX based system. Additionally, averaging helps reduce the effects of temporal resolution limits. Average cycle time or update period is inverted (i.e., update rate = 1/period) to yield the average update rate for that software component.

Measurement of cycle times for hardware external to the computer is also straightforward and can be accomplished by monitoring component signals (e.g., tracker serial line transmission, or CRT flyback scan) on an oscilloscope.

---

[1] We use the C callable function `gettimeofday` which, in our case, has a resolution of 0.833 msec.

2.3 Latency measurement

Measurement of elapsed computer execution time for specific activities (e.g., sensor data conditioning, model dynamics computation, etc.) within a given update cycle is accomplished by time stamp calls placed immediately preceding and following code sections of interest. Again, averaging is required to improve the reliability of stochastic data. Elapsed time measurement of this type is plausible only when source code is accessible; it is precluded for software libraries available only as object images.

Latency quantification for VE components that are external to the computer requires either relying on the veracity of manufacturers' specifications when available or on the development of calibration procedures specific to the hardware (e.g., spatial trackers [6]).

As discussed above, update and latency can be measured directly for system components external to the computer as well as for some of the code executed by the computer. A complete VE system timing picture can only be constructed once timing for all component *and* intercomponent communication is understood.

Measurements of communication timing between input devices and computer software or from computer software to output display by direct means is generally not feasible. For example, the time of data packet arrival at a serial port is easily detectable by monitoring the communication line on an oscilloscope. Measurement of the serial read delay—i.e., the time from when signals are present at the serial port until the packet's data is available to software in the computer—requires the event timer that detects activity inside the computer to be referenced to the time base of the oscilloscope that monitors external activity at the serial port. The most accurate measurement of internal computer activity on an oscilloscope, tapping directly into electrical signals on the computer's data or address buses, is also the most invasive and complex. A simpler noninvasive oscilloscope based method is to monitor the serial port for a write output generated by software in response to the availability of the data following the read. The second method, although less accurate because of the additional communication interface uncertainty introduced by the write function call, does provide an upper bound on the delay of the serial port read.

The noninvasive method, while less satisfactory for measuring delay across the interface from external hardware to internal software, is very well suited when the pathway to be timed spans external input hardware through internal software back to output display hardware, as is the case for a complete VE system. Additionally, once end-to-end latency and the timing of individual external hardware and internal software are known, the unmeasurable interface communication delays can be deduced. Thus, the end-to-end latency timing apparatus and procedures we describe next are based on this non-invasive strategy.

2.4 Latency Testing Apparatus and Procedure

The VE system pathway that we studied in this work is composed of one (or more) Polhemus Fastrak spatial tracker(s), a Silicon Graphics SkyWriter 4D/440IG2 computer, and a conventional 60 Hz interlaced CRT with NTSC timing protocol for output display. End-to-end latency was quantified by timing the delay between an input "motion event" and the event response visible on the CRT. The special apparatus for these tests follows equipment described by Mine [7] and includes a manually propelled swing arm[2] to impart motion to a Fastrak receiver, a slotted IR optical switch circuit (Figure 1a) mounted to the swing arm base, and a photodetector (Figure 1b) affixed by a clear suction cup to the front face of the CRT.

The Fastrak receiver, mounted at the distal end of the swing arm, sweeps out a circular arc section that is always within a 60 cm range of the tracker transmitter. Also affixed to the end of the distal end of the swing arm is a narrow (1 mm) tip extension which passes through the slot of the optical switch and mechanically interrupts the optical path between the photodiode (IR transmitter) and phototransistor (IR receiver) pair (see Figure 1a).

The latency testing software is based on a telerobotics VE application developed earlier in our laboratory [8]. The testing version of the software maintains all sensor input and computational capability of the original application, although most of this functionality is never used. The telerobotics VE consists of ~1000 untextured polygons. Images are rendered by the computer in a single window as a simultaneous red-blue stereo pair. The red and blue channels of the computer RGB output are

[2] The same swing arm described by Adelstein et al. [6] is used here, but with the electric motor not powered.

plugged into two separate monochrome monitors—equivalent to one for each eye in a stereo head-mounted display. In these tests we attach the CRT photodetector to only one monitor.

Prior to starting a series of tests, the Fastrak's report of swing arm position (in this case, the *y*-coordinate) at the point of interruption of the optical switch is set as a threshold in the testing software. Thereafter during the tests, crossing the threshold by moving the swing arm tip through the switch slot causes the appearance or disappearance of several white polygons on the CRT display. These respective polygon changes are sensed by the photodetector within its restricted field of view as black-to-white or white-to-black transitions. To maintain the polygons within view, position and orientation of the CRT output image are held stationary—i.e., the display window is not coupled to head tracker input.

Since the image presented on the CRT is scanned out sequentially from top to bottom of the screen, the local patch of screen under the photodetector lens is only illuminated when the electron beam sweeps by. Hence, photodetector output voltage in response to a white patch appears as a pulse train at the display refresh rate with individual pulse duration a function of CRT phosphor retention. When the patch is dark, photodetector voltage output remains flat. The appearance of the photodetector output waveforms (voltage V_2) for both black-to-white and white-to-black transitions can be seen in Figure 2.

The oscilloscope representation of optical switch and CRT photodetector outputs in Figure 2 also helps illustrate how the end-to-end VE latency is measured. A smooth rapid motion of the swing arm tip through the slotted switch produces a pulse in the phototransistor output voltage, V_1 whose rising edge triggers both the oscilloscope sweep and an external electronic timer (Tektronix Universal Counter model DC503). The transition of Fastrak output through the threshold in one direction causes the software to drive the CRT image from black-to-white (Figure 2a). In this case, the delay is visible on the oscilloscope as the time, τ_i^{bw} , elapsed to the rising edge of the first photodetector pulse. Time τ_i^{bw} for an individual measurement can be read directly from the DC503 counter to an accuracy of 0.2 msec or better.

Quantifying delay for white-to-black transitions arising from motion in the opposite direction is slightly more involved. Again the oscilloscope and electronic timer are triggered by the rising edge of the optical switch pulse in V_1. We note from Figure 2b that pulses in V_2 are visible on the oscilloscope only while the patch under the photodetector is white. The transition to black is marked by the absence of activity (symbolized by the dashed pulse) in V_2 after the last real pulse in the train. Therefore, the elapsed time in msec for a white-to-black transition for an individual measurement is given by

$$\tau_i^{wb} = \Delta_i^{wb} + 1000(N-1)/60 + 1000/60$$

where Δ_i^{wb} is the delay from the optical switch trigger to the first photodetector pulse as measured either by the oscilloscope or the electronic counter. The second term on the right hand side is the elapsed time between the N pulses in V_2 visible on the oscilloscope. The final term is the additional time for the CRT beam to sweep to the "absent pulse." The DC503 counter output permits $\tau_i^{wb} = \Delta_i^{wb} + 1000(N/60)$ to be computed to an accuracy of 0.2 msec.

Due to the latency variation encountered in a system with multiple asynchronous components, discussed earlier in Section 2, we make 25 (i.e., subscript i = 1, ..., 25) repeated measurements of both τ_i^{bw} and τ_i^{wb} by moving the swing arm back and forth through the switch location. The mean values of τ_i^{bw} and τ_i^{wb} are then combined and averaged to remove measurement bias from two physical sources. The first bias source is misalignment when originally centering the swing arm tip with respect to the optical switch's slot to set the Fastrak threshold value. The second is the small displacement difference (and therefore time difference) from the switch slot edges which cause the initial rise in V_1 that triggers the oscilloscope/timer to the slot center.

The resulting averaged bidirectional delay is also subject to the following measurement artifact that is caused by presence of a sampled data process in the VE system. We observe from the example data in Figure 3 that it is unlikely for a discrete time sample of a continuously varying signal to occur at the exact instant of threshold crossing. In effect then, for a hypothetical VE system without latency, the transition from black to white region could occur at up to one full update after the threshold was actually crossed and the optical switch triggered. Thus, the discrete time sampling process, by not detecting the exact instant of threshold crossing, adds a varying amount of delay that is equally distributed between zero and the effective VE update period, f_{eff} .

Because our concern is the data latency arising within the VE system, we need to remove from our measurements the artifactual contribution of time spent between discrete time samples waiting for new data to enter the VE pathway. Since the timing and position of Fastrak samples of swing arm trajectory are not available to our external instruments (i.e., the oscilloscope and universal counter), we cannot interpolate the exact instant of individual threshold crossings to correct individual latency measurements. However, since a number of measurements are being averaged, the expected value of the sampling artifact delay, equal to one-half the effective VE update period, $1/f_{eff}$, can be used for the correction.

One final latency measurement component that presents potential complication is the time to draw to a particular location on the CRT screen. The farther down the screen, the greater the time from start of the frame to rescan. In our set up, the CRT photodetector suction cup is located at the upper left corner of the screen and is estimated for interlaced 60 Hz NTSC image rendering to contribute an additional 2 msec to our end-to-end latency measurements. To eliminate CRT photosensor and location from consideration, we subtract the 2 msec from our measurements and include the VE latency only up until the data begins to be issued at the video port for scanout.

Consequently, the corrected estimate of end-to-end VE latency, τ_{ee}, from position detection by the tracker to first output at the CRT video port, is given in msec as

$$\tau_{ee} = 0.5\left(\frac{1}{m}\sum_{i=1}^{m}\tau_i^{bw} + \frac{1}{m}\sum_{i=1}^{m}\tau_i^{wb} - \frac{1000}{f_{eff}}\right) - 2.0$$

for m = 25 repeated measurements.

3. VE SYSTEM CONFIGURATIONS AND PERFORMANCE

The VE system hardware and software attributes common to all the tested configurations are outlined here. Details particular to individual system configurations are treated in the appropriate subsections.

As noted in Section 2.4, all work was carried out on an SGI SkyWriter 4D/440IG2 computer. The computer has four 40 MHz IP7 (R3000) CPUs and is equipped with dual Reality Engine-1 graphics pipelines (of which only one was used to generate the timing tests CRT images), two separate RS-232 serial port banks (i.e., separate UARTs), and, specifically for the final VE system configurations described below, a VME bus IEEE-488 parallel interface card (National Instruments model GPIB-1014-1). The operating system for these tests was IRIX 5.2.

For all tests, the Polhemus Fastrak(s) reported in 16-bit binary mode the absolute position and quaternion for each receiver—the most compact format for unambiguous position and orientation information. With RS-232 serial communication, this formatting results in 17 bytes per Fastrak receiver being sent to the computer for each measurement update.

The Polhemus Fastrak instrument (i.e., the control unit) measures and computes displacement for one receiver at a time and multiplexes between multiple receivers according to its internal 120 Hz update rate. Therefore, the two-receiver/one-instrument configuration has its update rate halved to 60 Hz per receiver[3]. A further implication of the multiplexing is that when the host computer application is operating on the most recent updates from two receivers, one receiver's data (the first one sampled) is actually 8.3 msec (1/120 sec) older.

All of our test software was written in C and uses WorldToolKit v. 2.1 (Sense8 Corp.) "tools" both to generate and run the VE. The WTK tools comprise both a simulation "framework" that the application must use and a library of C callable functions that includes graphics and mathematics utilities and drivers for a variety of commercially available input sensors. The WTK simulation framework requires that a "universe" be created first before the simulation is run. The creation phase usually includes loading or building graphical objects, setting lighting and viewing parameters, specifying sensors, and initializing the simulation. During the run phase, WTK operates as a *single* process that repeatedly loops through a series of tasks. These tasks include WTK functions that handle many common simulation chores such as reading sensors and rendering the scene. WTK also allows for the execution of programmer (non-WTK) specified functions that govern the actual interaction

[3] Note that these rates are the maximum the Fastrak instrument itself can achieve. The rate apparent to the application will be lower if data communication or the application cannot keep pace.

and dynamics specific to the particular VE (e.g., manipulating sensor data and checking for threshold crossings in our latency testing application).

The Fastrak driver provided by Sense8 is not used in any of our work. We wrote our first Fastrak driver (the same one used in the baseline configuration) following the serial interface specifications for WTK because, initially, Sense8 did not offer one for the Fastrak. We continued to use the our driver because of the additional flexibility and control it afforded over the later Sense8 version. The subsequent Fastrak drivers that we have developed in the course of this work are very different than the WTK tracker driver concept.

3.1 Original (Baseline)

3.1.1 Configuration

The original configuration was based on the single process paradigm used by WTK in which the Fastrak is operated in "polled" mode—i.e., the tracker instrument only returns data to the host computer in response to a prompt from the application. The single application cycle is a series of tasks in the WTK loop, structured in the "Read-Poll-Simulate-Draw" (RPSD) ordering commonly employed for "real-time" graphics.[4] The "read" in the current cycle is carried out on data that was polled from the Fastrak in the previous cycle. Immediately after successful completion of the current read, the "poll" is sent to the Fastrak instrument with the goal of having data arrive in time for the next cycle's read. Because the Fastrak then processes the data request separately from the computer, the remainder of the application cycle can proceed to the "simulate and "draw" segments without waiting. We use the term "simulate" to delineate the accessible portion of the application where programmer specified and WTK functions manipulate the contents of the most recent sensor data. The "draw" segment is where the inaccessible internals of WTK render the images. The double buffering swap call (IRIX function `swapbuffers`) is the last action in the draw segment handled by WTK before returning to the top of the application loop.

The baseline configuration employed a single poll and read to retrieve data from the two receiver Fastrak instrument. All RS-232 communication between the Fastrak and computer occurred at 19.2 Kbaud, corresponding to an 8.85 msec transfer time for the individual receiver's 17 byte position and quaternion packet. Because the transfer time per packet exceeded the instrument's internal 8.3 msec multiplexing period, the two receiver data stream, as seen on an oscilloscope, coalesce into a contiguous 17.7 msec packet.

3.1.2 Performance

Because data from the two Fastrak receivers is measured and transferred sequentially in the baseline configuration, we used separate swing arm tests to examine the delay pathway from each receiver through to the CRT. The first sensor (s1)—the one whose data is processed and delivered first during the update cycle and therefore has greater age by the time the simulate phase of the application begins had a corrected latency, τ_{ee} = 64.0±5.2 msec.[5] The second sensor (s2) had latency of τ_{ee} = 54.1±5.6 msec. This latency difference between the two sensor paths is in good agreement with the 8.3 msec expected due to receiver multiplexing by the sensor.

The average update rate for the baseline configuration, measured by comparing 1500 repeated time stamps at the top of the loop for two separate tests, was 20.3 Hz (update period mean±stdev = 49.2±10.6). Figure 4 breaks down the 49 msec application loop time into average times of 35, 1, 1, 3, and 9 msec respectively for the read, poll, convert, simulate, and draw cycle components obtained from internal time stamps. The "convert," added to the RPSD model presented above, accounts for various conversions and scaling performed on the raw binary Fastrak data. After the poll portion of the code is complete, the average time until the start of the read at the beginning of the next (j+1) cycle is only 3 + 9 = 12 msec.

Oscilloscope observations of the serial lines carrying the poll request to and the returning data from the Fastrak, indicate a delay ranging between 7 and 24 msec before the Fastrak begins to send data back to the computer. Polhemus [9] indicated that

---

[4]The less preferred alternative to RPSD ordering, "Poll-Read-Simulate-Draw" (PRSD) must sit idle in the loop waiting for all sensor data to return before being able to complete the read and proceed with the simulate and draw portions of the application. The consequence of the longer loop time is a slower update rate for PRSD.

[5] Mean ± stdev calculated from m = 25 repeated measurements.

the 7 msec is the time for the Fastrak interrupt service routine to respond to the poll request. The remaining zero to 16.7 msec variability (8.3 msec average) corresponds to the asynchrony of the poll from the host computer with respect to the Fastrak's internal clock—i.e., data can only start being sent at a particular point in measurement cycle. Summing the 0.5 msec for serial transmission of the single character pole command, the average 15.3 msec until the Fastrak begins its serial line response, plus the 17.7 msec to return the full 34 byte packet at 19.2 Kbaud, yields an average response time of 33.5 msec. Thus, on average, the response time of the Fastrak exceeds the simulate and draw time of the application by 33.5 – 12 = 25.5 msec and lingers on into the next (j+1) application cycle. We presume then, that by trimming away these 25.5 msec, a major portion of the read time will be eliminated, thereby shortening the loop cycle time and increasing the application update rate.

3.2 Forked-Off Fastrak (FOFT)

3.2.1 Configuration

From internal time stamps in the application cycle and oscilloscope observations of Fastrak serial line communication, we were able to identify a cause of the baseline configuration's slow update rate. The principal remedy proposed for this next configuration, Forked-Off Fastrak (FOFT), was to remove communications with the Fastrak from the application loop by developing a stand alone driver process to handle tracker data.

FOFT was our first attempt at separating the tracker process from the application process. It was our intention to make the driver software process run at a high priority on its own processor, so that the computer would always be ready to receive the data, and then the data would be the freshest possible. The method we used to create two processes was to have the application map and initialize shared memory, and then fork into two processes. The child process would then initialize and run the Fastrak and write its data into shared memory. The parent process would initialize and run the WTK simulation, read the tracker data from shared memory, and use the data in simulation.

While the separate tracker driver process would be expected to speed up the application, the driver process itself is still limited by the long response time of the Fastrak to polled data requests. Therefore, to shorten the tracker driver cycle, FOFT allows for continuous Fastrak output at 38.2 Kbaud in addition to the polled mode 19.2 Kbaud data transfer used in the baseline configuration. In continuous mode, the Fastrak does not wait for prompts from the host—it sends out data as soon as they are ready at the maximum rate permitted by its internal clock and data transmission line. FOFT still relies on a single read (and a single poll when in polled mode) to retrieve the combined 34 bytes data sequence from the two receiver Fastrak instrument.

In the hope of tightening up some of the variability associated with a UNIX environment, we also investigated the effects on the driver process of a number of the "real time" extensions which are available in IRIX 5.2 as arguments to function `sysmp`. The tested extensions allowed locking a process into memory; setting a maximum, non-degrading process priority; locking the process onto a processor (CPU 3 was used for the tracker driver); restricting the processor from running other processes; isolating the processor; and moving the system clock to CPU 0.

3.2.2 Performance

Latencies and update rates were measured and compared for a variety of FOFT setup conditions. These conditions include polled versus continuous mode Fastrak output; 19.2 versus 38.4 Kbaud serial line transmission rate; presence versus absence of real time extensions; and, again, first (s1) versus second (s2) receiver delay.

The basic FOFT implementation, in which the tracker still operates in polled mode and transfers data at 19.2 Kbaud, saw the update rate for the application process rise to 58.1 Hz (update period = 17.2±2.4 ms). However, the update rate for the new tracker driver process remained at the 20.3 Hz (update period = 49.3±9.8 ms) measured in the baseline configuration. In terms of effective update rate, f_{eff}, the VE system remains at 20.3 Hz, the update time of the slowest component in the pathway. Furthermore, we observed that the end-to-end latency for the second receiver's (s2) pathway grew to τ_{ee} = 67.3±7.3 msec. Including the real time extensions in the basic FOFT implementation had no discernible effect on either application and driver rates or s2 pathway latency.

Switching to the Fastrak continuous output mode has a decisive effect on system update rate. While, the application update rate is unchanged at 57.9 Hz (period = 17.3±2.8 msec), the driver cycle and therefore the effective update rate, f_{eff}, jump to

50.8 Hz (period = 19.7±29.7 ms). End-to-end s2 latency, however, decreased only slightly from the polled FOFT implementation to 63.1±7.5 msec—which is still worse than the level achieved in the baseline configuration. Because the real time extensions also had no obvious impact on continuous mode operation, their inclusion was not considered in any subsequent system configurations.

One intermediate observation from the 19.2 Kbaud continuous FOFT tests was the inability of the Fastrak, and therefore the tracker driver, to achieve a 60 Hz update rate. Because the 8.85 msec time to transmit a full 17 byte packet exceeds the internal 8.3 msec clock cycle at which newly processed tracker measurements become ready, the new data must wait until old packet has been completely transmitted. This 0.5 msec slip accumulates until reaching a maximum value at which point the Fastrak stops transmitting and discards data that are already in the serial queue [9]. The Fastrak then resynchronizes by waiting until its next internal clock cycle to restart the transmission with new data. The proportion of data transmitted relative to the one discarded frame corresponds to the reduction observed in tracker update rate.

Increasing the serial line to 38.4 Kbaud—if feasible within serial cable length constraints—cuts the transmission time per receiver packet to 4.4 msec. This ends the data overrun problem and results in an increased driver process update rate to 56.7 Hz (period = 17.6±27.8 msec). The application continues to operate at the frame rate as before: 57.8 Hz (period 17.3±2.2 msec). Thus, the effective VE update rate, f_{eff}, rises to 56.7 Hz. Because of the faster serial transmission time and the absence of data slip, end-to-end latency, τ_{ee}, for s2 drops to 55.3±7.3 msec which approaches the original baseline configuration level.

The s1 versus s2 receiver path latencies were compared for continuous mode output at the two baud rate settings. The latencies, τ_{ee}, measured for receiver s1 were 74.1±7.2 and 65.1±8.1 msec respectively for 19.2 and 38.4 Kbaud serial communication. These were longer than the comparable s2 values reported above by an amount consistent with the internal Fastrak multiplexing cycle.

The average internal timing shown in Figure 5 is typical for the application, tracker driver, and shared memory segments of all continuous FOFT implementations regardless of baud rate or inclusion of real time extensions. Figure 5 also represents accurately the application portion of the polled FOFT implementations.

A new component introduced by FOFT's dual process organization is the time that data written into shared memory by the driver process must wait until fetched by the application process. The average 8.3 msec spent in shared memory by FOFT is close to the theoretical expected value for an equiprobable distribution of random waiting times between two completely asynchronous processes with the driver and application processes' update rates. Based on the measured update times reported above for FOFT, a major source of the asynchrony is likely the high variability (i.e., stdev much greater than mean value) in driver cycle period.

3.3 Asynchronous Serial Tracker (AST)

3.3.1 Configuration

The main drawbacks of FOFT had to do with its structure. Because of the forking procedure, the application and tracker driver, though separate processes, were still tied together. Thus, if the application exited or crashed, it usually also brought down its child process—i.e., the tracker driver. Conversely, certain tracker errors could cause the simulation to exit. This made FOFT overly complicated, and the tracker portion of the code hard to debug. The Asynchronous Serial Tracker (AST) configuration described in this section truly separated the two processes. AST, the actual tracker driver process, could be debugged and run independently from the application.

Following the configuration of FOFT, AST was developed initially as a single driver process to be used in conjunction with a single Fastrak instrument and two receivers. In the final AST configuration, multiple AST tracker driver processes can operate simultaneously and asynchronously, each with its own Fastrak instrument and up to four receivers per instrument.

The configuration used for testing AST performance consisted of two Fastrak instruments, each with a single receiver. With this arrangement, we sought to avoid multiplexing between multiple receivers and have the trackers and drivers approach the maximum achievable 120 Hz instrument update rate. The advantage of eliminating multiplexing would be receivers with

equal length latency paths. The faster update rates would also theoretically reduce the magnitude of interprocess asynchrony and, consequently, the interprocess communication latency due to data waiting in shared memory.

AST processes can be started at anytime prior to running the application and can be kept running even after the application has been stopped. AST first uses software keys to map sections of shared memory (one for each sensor). AST then initializes the Fastrak and runs in continuous serial transmission mode at either 19.2 or 38.4 Kbaud, storing each receiver's data in the appropriate shared memory area. The application maps to the same areas in shared memory areas and is therefore able to retrieve tracker data for used in the simulation.

Each shared memory area contains several fields. The position and orientation data areas are double buffered so that buffer reading and writing can occur simultaneously without corrupting the data currently being read by the application. The shared memory areas have flags to tell which buffer has the freshest data and whether a buffer's data has been read before, read and write locks, as well as counters to indicate the current driver cycle number. In addition, two more flags are used: one to signal when the driver should exit and one for general purpose communication.

We also wrote a shared memory monitoring program (SMMON) to help debug and control the tracker driver. SMMON can map to any of the shared memory areas to monitor the data. It can read all fields, and set or clear the flags area of the shared memory to allow communication between the monitor, driver processes, and the application.

3.3.2 Performance

AST performance is reported here for 38.4 Kbaud data transfer. Under these conditions, there is no serial transmission slip so data is never discarded and the Fastrak instrument and driver both update at the maximum attainable rate of 120 Hz (period = 8.3±8.5 ms).[6] Thus, the application cycle's rate, which was measured at 53.6 (period = 18.7±3.0 msec) and 54.9 Hz (period = 18.2±2.8 msec) during separate tests, now becomess the restraint on effective VE update rate, f_{eff}.

End-to-end latencies for the s1 and s2 pathways, measured at 50.7±7.5 and 50.4±7.7 msec respectively, indicate that the elimination of multiplexing by using two separate single receiver Fastrak instruments produced the expected result of equal delays between multiple receivers. Also, these values for τ_{ee} finally undercut the shortest latencies measured in the original baseline configuration.

The averaged internal time stamping data for the AST configuration are provided in Figure 6. The presence of two separate AST drivers, shared memory areas, and fetches is denoted by the (0,1) suffixes. Both drivers and shared memory areas, however, exhibit the same timing characteristics.

We note that the average 8.3 msec in shared memory with the higher AST tracker driver rates is no different than that obtained with FOFT and shows the absence of any reduction in magnitude of the randomly distributed asynchrony between the driver and application processes, even with the much improved driver update rate. The high degree of asynchrony is not surprising considering the wide variation in measured AST driver update period (8.3±8.5 msec) noted above.

3.4 Single Parallel Interface (SPI)

3.4.1 Configuration

The Single Parallel Interface (SPI) configuration was developed to reduce end-to-end latency by replacing RS-232 serial communication with the high speed (100 KByte/sec) IEEE-488 parallel interface that is standard on the Polhemus Fastrak. At the 100 Kbyte/sec IEEE-488 interface rate permitted by the Fastrak, a complete receiver data packet (now 19 bytes long to accommodate extra IEEE-488 protocol trailing information) is transmitted in less than 0.2 msec.

---

[6] One curious observation during early debugging of AST was that when two Fastrak instruments communicated with two serial ports that were on the same computer serial board, driver update rates dropped precipitously. The update rate could only be brought up to the maximum rate by connecting one of the instruments to a port on a second board. We presume that this is because serial ports on the same computer board share the same UARTs and must therefore be read sequentially and therefore less frequently. With separate UARTs available on two boards, the two serial ports can be read simultaneously.

The National Instruments model GPIB-1014-1 board, inserted into our SkyWriter's VME bus, served as the computer's IEEE-488 interface. SGI furnished us with the software subsystem for the GPIB-1014. This subsystem, which includes the device driver and National Instruments' GPIB software function library (NI-488M), is not normally included in the IRIX 5.2 distribution.[7]

SPI permits the use of only a single Fastrak instrument with up to four receivers. The SPI driver, however, continues to read tracker data one receiver at a time (as opposed to the two receiver data packet read of FOFT). It also maintains the shared memory scheme of AST, keeping the same separate areas of shared memory for each receiver's data. Thus, a single version of the application can retrieve data placed in shared memory by either AST's or SPI's tracker driver.

3.4.1 Performance

SPI performance was evaluated for a two receiver configuration. Application update rates of 60.0 Hz (period = 16.7±1.2 msec) and driver rates of 60.1 Hz (period = 16.7±0.4 msec) were measured, yielding an effective update rate of 60.0 Hz. Of note in these measurements is the near elimination of driver cycle time variability with SPI. We conjecture that the cause of this improvement is the responsiveness of the GPIB-1014 hardware, driver and function library when compared to the "termio" subsystem developed over the years for the UNIX handling of asynchronous I/O devices such as serial port terminals.

End-to-end latencies of τ_{ee} = 43.2±6.7 and 41.4±6.8 msec respectively were measured for the s1 and s2 paths. Internal latency components are given in Figure 7.

While the average shared memory time is again 8.3 msec, the time distribution is no longer purely random. It has the repeating ramp pattern shown in Figure 8 for each of receivers s1 and s2. The ramping increase (between periodic resets) in shared memory time for a given receiver is characteristic of a slip in synchronization between a faster input (i.e., driver) cycle time and a slower output (i.e., application) cycle time. The ~8.3 msec difference between s1 and s2 times apparent in any cycle in Figure 8 is due to the application fetching multiplexed single instrument readings. Furthermore, because of the 8.3 msec difference between receivers, whenever the slip pattern restarts after hitting a maximum, the role of "oldest" and "newest" receiver is reversed. This periodic reversal between oldest and newest receiver explains why the τ_{ee} values listed above for the s1 and s2 pathways in SPI are approximately the same.

3.5 Enhanced Serial Parallel Interface (SPI+)

The enhanced Serial Parallel Interface (SPI+) configuration incorporates two simple modifications to SPI that result in further reductions of end-to-end latency.

The first modification exploits the ramp pattern for shared memory time presented in Figure 8. We note that if a shared memory time threshold is set such that data retrieved by the application cannot exceed this maximum age, the amount of slip permitted to accumulate will be reduced. The smaller slip would therefore decrease the shared memory contribution to overall latency. The limitation to this approach is that thresholding can only be applied to one receiver's shared memory time. The second receiver, as shown in Figure 8, which is synchronized with respect to the first must spend 8.3 msec longer in shared memory (that is assuming the threshold is set to 8.3 msec or less).

In tests of SPI+ with a shared memory maximum age threshold of 4 msec on receiver s2, we observed a large drop in s2 pathway latency to 33.5 msec. The application rate, and therefore, the effective update, rate fell slightly to 59.5 Hz. We assume that this drop was due to the discarding of occasional "old" samples already in shared memory when the age threshold was exceeded.

The second modification may have less general impact beyond tightly controlled applications such as our test VE in which the content of the rendered scene is not varied broadly enough to induce wide fluctuations in the application cycle time. We observe from the internal timing for the series of VE system configurations in Figures 4 through 7 that the "draw" portion of

[7] SGI has stated that there will not be a driver for this board beyond IRIX 5.2. In subsequent IRIX versions, SGI drivers will only be available for National Instrument's SCSI bus GPIB interface.

the application can take different amounts of time, even though the images are the same in all cases. As described in Section 3.1.1, the draw portion of the application is the inaccessible rendering portion of the WTK code that is only completed after the buffer swap. Since this buffer swap is synchronized with the computer video hardware frame refresh, we propose that when the draw portion of the code takes longer, it is because the application is sitting idle waiting for that refresh before looping back to the top of its cycle.

We tested this idea by inserting our own time controlled stall in the SPI application code. We observed that as much as 4 msec of idle time could be added to the application (when not thresholding shared memory data age) without degrading the update rate. By placing this stall segment to the top of the loop (see Figure 7), the idle time originally in the draw segment is now consumed before tracker data is fetched from shared memory, and is therefore removed from the VE latency path. For SPI+ conditions of 4 msec of stall combined with a 4 msec shared memory age threshold, an end-to-end latency of τ_{ee} = 30.7 msec combined with an effective update rate of f_{eff} = 57.5 Hz.

4. CONCLUSIONS

In this paper we have presented a series of hardware and software reconfigurations that in the final implementation has significantly improved end-to-end latency and update rate in our UNIX based VE system. The significant steps in the development of our current VE configuration (including the stage at which they were introduced) were:

- Separation into separate tracker driver and application processes using shared memory data transfer (FOFT).
- Continuous mode tracker data transmission (FOFT).
- Multiple concurrent tracker driver processes (AST).
- High speed IEEE-488 parallel interface tracker interface (SPI).
- Shared memory data age thresholding (SPI+).

Throughout each stage of this work we used objective, detailed performance measurements to guide subsequent development. The best end-to-end latencies and effective update rates measured for each configuration are summarized in Figure 9.

Many of the results and techniques developed in this work are transferable to other VE systems. The measurement apparatus and methods can be easily replicated and applied to quantify VE behavior in any system with a tracker and available CRT display. The series of hardware and software configurations that we studied can be implemented on most UNIX based VE systems.

5. REFERENCES

[1] R. Azuma and G. Bishop. "Improving static and dynamic registration in an optical see-through HMD. *SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series*, pp. 197-204, 1994.

[2] M. Deering. "High resolution virtual reality." *Computer Graphics*, Vol. 26, pp. 195-202. 1992

[3] M. Friedmann, T. Starner, and A. Pentland. "Synchronization in Virtual Realities." *Presence: Telepresence and Virtual Environments*., Vol. 1, pp. 139-144, 1992.

[4] J. Liang, C. Shaw, and M. Green. "On temporal-spatial realism in the virtual reality environment." *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology* , pp. 19-25, 1991.

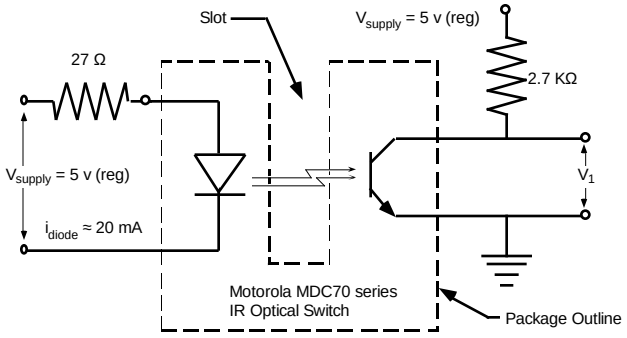
[5] K. Zikan, W.D. Kurtis, H.A. Sowizral, and A.L. Janin, "A note on human head motions and on predictive filtering of head-set orientations." *Proceedings of SPIE: Telemanipulator and Telepresence Technologies*, Vol. 2351, pp. 328-336, SPIE-The International Society for Optical Engineering, Bellingham WA, 1994.

[6] B.D. Adelstein, E.R. Johnston, and S.R. Ellis. "Dynamic response of electromagnetic spatial displacement trackers." *Presence: Telepresence and Virtual Environments.* In press.

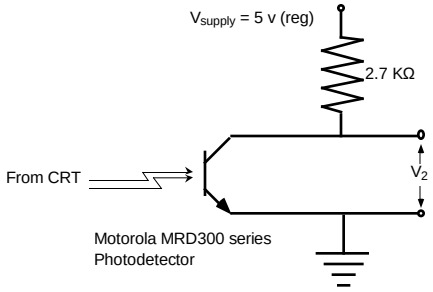
[7] M.R. Mine. *Characterization of End-to-End Delays in Head-Mounted Display Systems* (Tech. Rep. TR93-001). University of North Carolina, Department of Computer Science, Chapel Hill, NC, 1993.

[8] R. H. Jacoby and S. R. Ellis,. "Using Virtual Menus in a Virtual Environment," *Proceedings of SPIE: Visual Data Interpretation,* Vol. 1668, pp. 39-48, SPIE-The International Society for Optical Engineering, Bellingham WA, 1992.

[9] Personal Communication. John Brown, Polhemus, Inc. September 23, 1994.

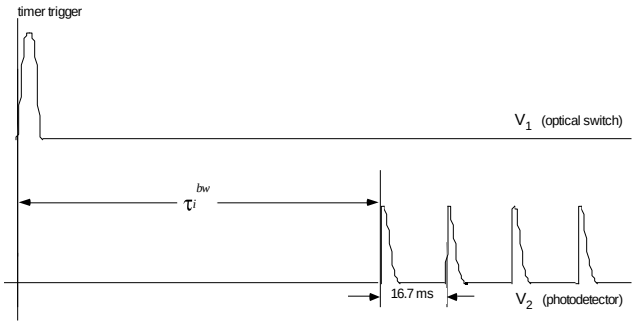


(a)

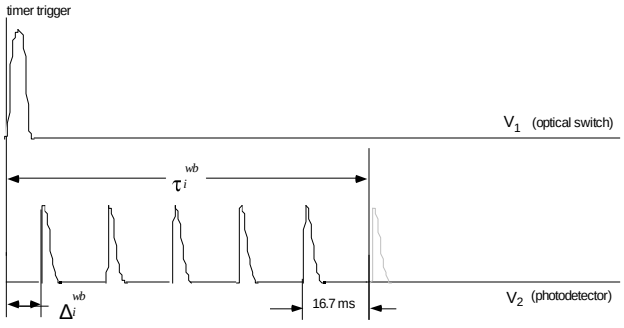


(b)

Figure 1. (a) Swing arm optical switch. (b) CRT photodetector.



(a)



(b)

Figure 2. Data appearance on scope. (a) Black-to-white photodetector transition. (b) White-to-black photodetector transition.

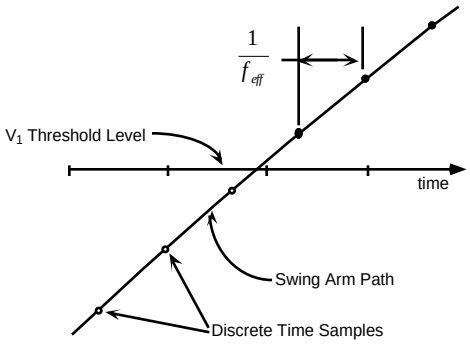


Figure 3. Sampled VE data for a continuous swing arm motion. The circles represent discrete data samples that have propagated through the VE system to the CRT display. Filled circles correspond to Fastrak values for which the CRT photodetector detects black; the hollow ones correspond to the white condition.

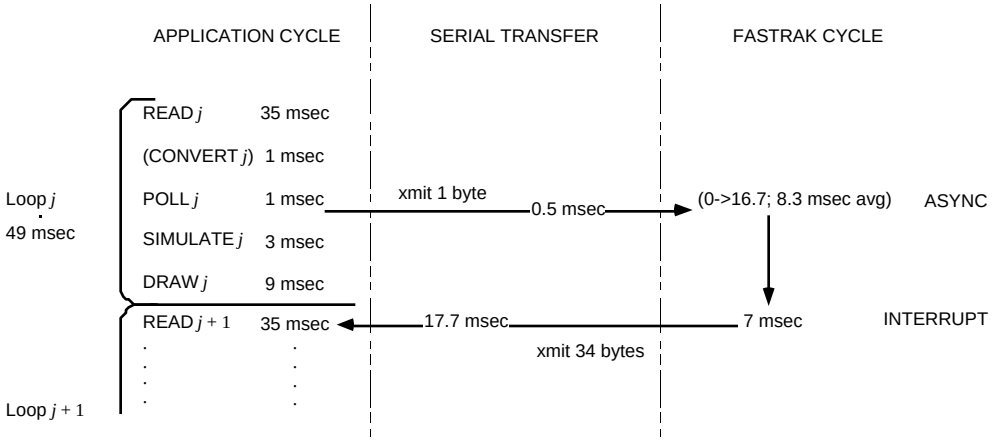


Figure 4. Original configuration internal time stamping.

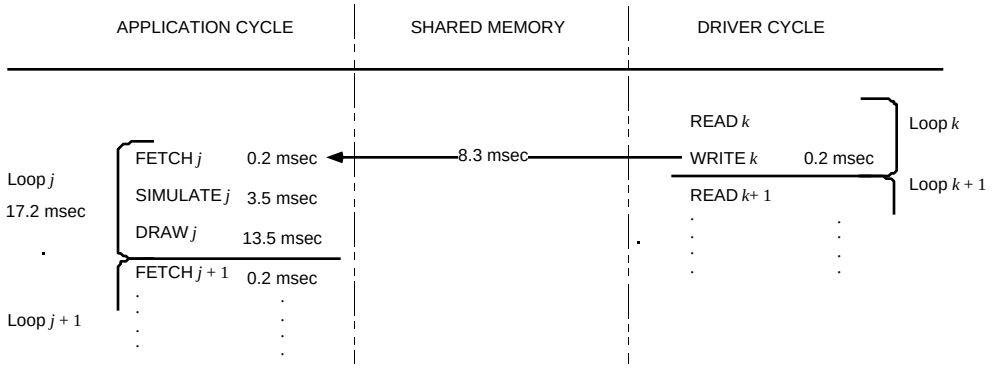


Figure 5. FOFT configuration internal time stamping.

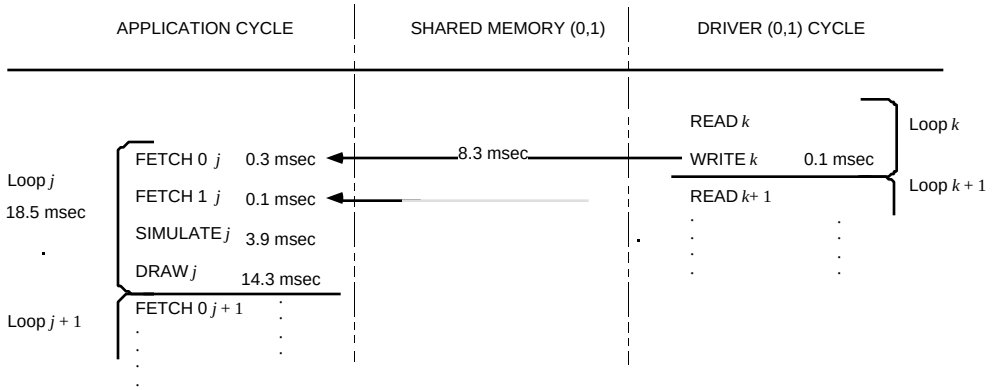


Figure 6. AST configuration internal time stamping.

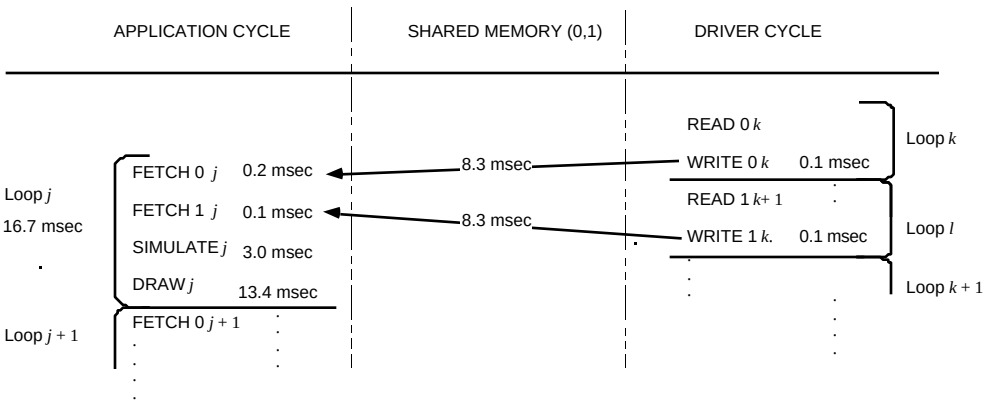


Figure 7. SPI configuration internal time stamping.

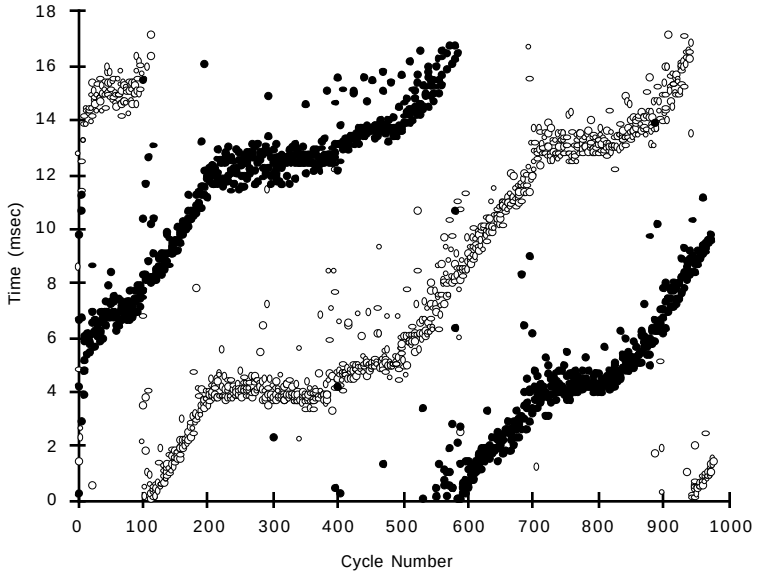


Figure 8. SPI shared memory data slip for s1 (filled circle) and s2 (hollow circle) receivers.

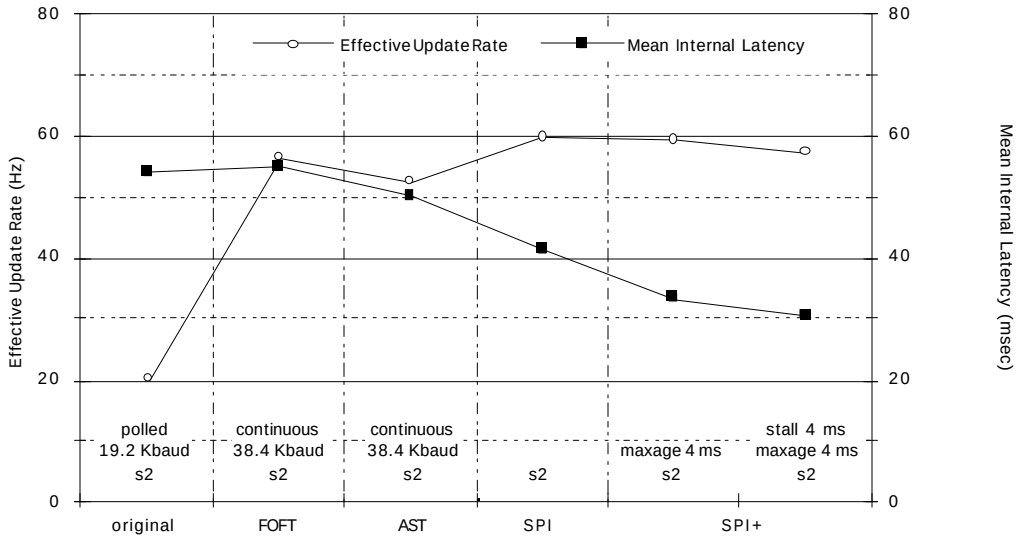


Figure 9. Measured end-to-end latency (τ_{ee}) and effective update rate (f_{eff}) for different system configurations. Plotted data represent the best performance achieved for that configuration.