A Simple Method for Estimating the Latency of Interactive, Real-Time Graphics Simulations

Anthony Steed*
Department of Computer Science, University College London

Abstract

One of the critical determinants of the effectiveness and usability of interactive graphics simulations is the latency with which visual updates can be made based on input from interaction devices. High latency can diminish performance and can lead to simulator sickness. We demonstrate a new method for measuring latency using a standard video camera. The method is simple to configure, sensitive and rapid to use. This is in contrast to previous methods which required specialized equipment, were laborious or could only determine gross changes in latency. We attach a tracker to a pendulum and move a simulated image on the screen using the tracker positions. We video both the pendulum and simulated image together, and fit two sine curves, one to centre of motion of pendulum and one to the centre of motion of the simulated image. From the phase difference between these two sine curves we can determine latency changes significantly less than the frame rate of the camera. We demonstrate the method by comparing the latency of a two different systems for a CAVE™-like display.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.3.2 [Computer Graphics]: Graphics Systems

Keywords: Latency, Performance, System Design, Real-Time Graphics, Interactive Systems

1 Introduction

Real-time simulations are greatly enhanced by the fluidity of their interactions. Fluidity refers to a combination of the update rate of the displays, and the end-to-end latency of the system. The end-to-end latency is the time taken from an input device changing state to a consequent change on the screen. The update rate of the simulation is mostly dependent on the application complexity and rendering performance of the simulation. Although the end-to-end latency critically depends on the update rate of the display it additionally depends on several other issues, including the type of input devices, types of serial and network connection, operating system and application threading structure. Today many installations will use PC networks for rendering, rather than single machine renderers. This means that the end-to-end latency can be hard to predict because there might be several CPUs and more than one type of network or device interface. The relationship between latency and update rates, and the potential sources of latency are discussed in more detail in Section 2

Latency and update rate have both been shown to impact task performance or user response to the simulation (e.g. [Bryson and Fisher 1990; Ellis et al. 1999; Ellis et al. 2002, Meehan et al. 2003]). Latency can also cause nausea and simulator sickness [Craig et al. 2000]. Monitoring latency so that it can be combated is thus an important part of the implementation and maintenance of real-time simulations.

In this paper we present a method for estimating the end-to-end latency of a graphics simulation system. We measure latency by simultaneously capturing a video of a target attached to a tracking device moving on a pendulum, and the movement of a simulated image that uses the resulting tracking data. By fitting two sine curves, one to the horizontal displacements of the tracker, and one to the horizontal displacement of the simulated image, we can accurately track the phase difference of the two, and thus the latency of the complete system. The contribution of the paper is that our latency estimation technique combines the use of off-the-shelf hardware, simple setup, automatic latency estimation given an appropriate video sequence and latency estimation with accuracy at a resolution finer than the frame rate of the video

We demonstrate the method by comparing the latency of two different systems for driving a CAVE™-like system [Cruz-Neira et al 1993]. We show that we can accurately measure known latency offsets and known introduced delays in the system. We also show that the method is more accurate than a standard frame counting method with a high frame rate video camera.

2 Related Work

2.1 Sources of Latency

Latency in interactive real-time graphics simulations comes from various sources [Mine 1993]:

- sensor reading and computation,
- sensor data communication
- application computation
- rendering computation
- display refresh.

Any particular installation may have extra complexity at several stages. The tracking devices might be plugged in to a dedicated machine that acts as a server on the network so that many applications might connect. Although application simulation and rendering might be done on a single PC, for large-wall or CAVE™-like systems the rendering might be done on a separate PC-based cluster connected to the application node via dedicated networking. The rendering cluster itself might have a frame buffer integration stage such as sort-last rendering. On a high-end

*A.Steed@cs.ucl.ac.uk

rendering system, the rendering itself might be multi-processor or multi-stage procedure.

Some aspects of installed systems are usually fixed. Most commonly the tracker device will have a known, fixed sample rate and the display system will have a fixed refresh rate. In contrast, the application and rendering compute times will vary between simulations and may vary within a simulation.

Aspects of the system installation that are difficult to control are the timing of activities that transfer data between machines or processes. With 1GB Ethernet now commonplace, in many facilities it is common to host interface devices off dedicated machines rather than having application software interface directly to the interface device connected to the same host. Although the theoretical latency of networked systems can be low, in practice latencies of processes that are scheduled on device or network input or output depend critically on the timing of the signals that trigger the input or output.

As example of the potential issues in system configuration, consider a tracking device that can output data at 60Hz. That is, every 16ms, it becomes ready to send data on a serial or USB link. To get the lowest latency, the application process must simultaneously be ready to receive this data and exploit it. To achieve this, one could set up the tracking device to send data continuously, and then time the application so that it finished rendering one frame at the same time that new data was available so that the application could start work on the next frame directly. In practice this would require that the both application simulation and display ran at 60Hz, and could be synchronized. Synchronizing the processes so is not commonly done in practice, except on integrated hardware such as games consoles, so typically a separate process would read data in to memory, with the application simply taking the latest values out of memory. If the application also runs at 60Hz, this could incur an overhead of up to 16ms latency each frame. We might expect 8ms on average, but we could persistently get 16ms if we were unlucky. A higher data read rate obviously reduces the problem, though it does not eliminate it.

Continuing this example, we could then note that 60Hz is not a common refresh rate for high-end displays such as LCDs or projection walls. A typical LCD monitor might run at 72Hz, and if the rendering runs in less than 13.8ms, the graphics update rate could be 72Hz. This means some frames will be rendered with the same tracking data, but more importantly, the tracking will get out of phase, so that some frames will be rendered with tracking data that could be up to 16ms out of date. Thus the latency of a system might vary from screen refresh to screen refresh, and it is would be necessary to sample the latency several times to get a reliable estimate. As a concrete example of these issues Jacoby et al. [1996] discuss the latency improvements that are possible by changing how a Polhemus Fastrak is used to connect to a single-machine simulator. By moving from the default device configuration, application and cabling to a revised one, the end-to-end latency was reduced from 65ms to 30ms.

We note that although we are discussing end-to-end latency of the system, we need to be precise about what we are actually measuring. For example, different tracked devices on the same tracker hardware might be reported at different timings simply because of the way the tracking device itself works. Latency might be measured with a single tracked device, which could be lower latency than the normal use when two or more tracked devices are used. Also, there might be variation in display response. The top left of a CRT is shown before the bottom right, so the latency response will be a few milliseconds different. With an LCD display, the screen typically refreshes at the same time, but different colors and different brightness levels take longer to stabilize. With a multi-display system, different display cards, and even different monitors on a single card might generate images at different times unless all the images are genlocked or frame-locked together. Thus there is rarely a single latency for a system and we need to be very specific about what we might be measuring.

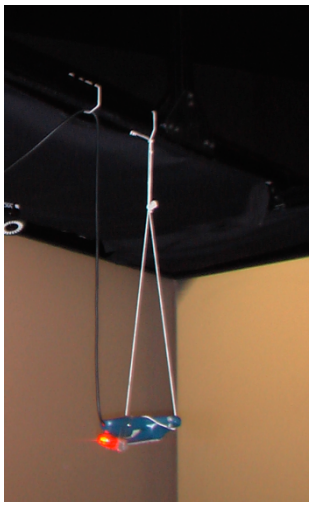

Figure 1: A wired Intersense IS900 controller used as pendulum with two attachment points. A red LED is attached.

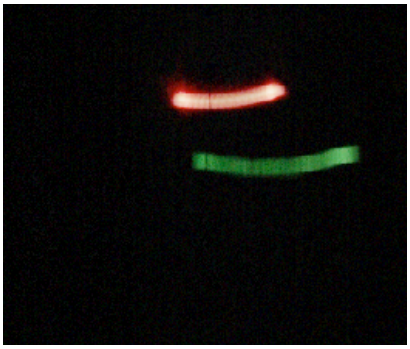

Figure 2 A summary image of a typical video capture, with red (upper line) being the physical pendulum and green (lower line) the simulated image. The image is created by taking the mean color across the video sequence minus the median color across the video sequence, and then normalizing in each color channel.

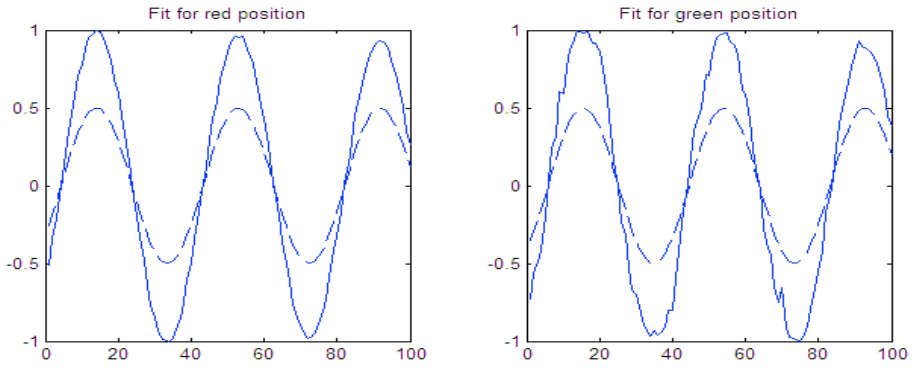

Figure 3: Fit of red (physical pendulum) and green (simulated image) displacements in a video 100 frames (4s) long. The solid lines indicate the normalised displacements, and the dashed curves, the fitted sine curve. For clarity the fitted sine curves are scaled to half their amplitude.

2.2 Measuring Latency

Because latency is such an important issue several previous estimation methods have been described. The most commonly reported in use (e.g. [Meeham et al. 2003]) is Mine's method [Mine 1993] which uses an oscilloscope to measure the time between the firing of two photo-diodes, one on a pendulum which passes a small light, and another that registers the flash of a pixel on a screen. Because such systems take measurements at fixed physical positions, they need careful calibration to ensure that they are measuring the same position on the swing of the pendulum and the graphical representation. Adelstein, Johnston and Ellis [1996] present a more technically sophisticated arrangement with a driven pendulum that they use to analyze the latency of several trackers. Both these methods require some specific hardware or infrastructure to be built.

Liang, Shaw and Green [1991] count frames in a video at 60Hz of a tracker attached to a moving pendulum. From tracking data, they know when the pendulum passes the vertical, and by recording tracking data via the video image, they can deduce the latency in the number of frames. This particular method requires reconfiguration of the tracker space, which is impractical in some situations. Swindells, Dill and Booth [2000] present a similar, but more accurate method that uses a tracker attached to a driven turntable. They create a virtual turntable using tracking data, and by capturing video of the real and virtual turntables together, they can estimate latency by the angular difference. A simpler method using a camera is described by [He et al. 2000]. A video is made of a tracker moving in front of a display that shows an object controlled by that tracker. By counting the number of frames between noticeable turning points in the motion, one can get an estimate of latency. However this is a laborious method, and prone to error in the identification of the turning points in motion. We term all of these types of approach, *frame counting methods,* because their temporal resolution is in multiples of the video rate. With sufficient samples they can be accurate, but some of the methods are sensitive to the fact that they only take periodic samples, whereas latency can vary quickly.

Miller and Bishop [2006] note that with a motion detection algorithm, one can determine the frame offset of a motion automatically, but again the latency is detected in multiples of the frame rate.

In contrast the method we propose estimates latency with a normal video camera and the estimation is automatic once the video is captured. We show the method is more accurate than the frame counting method with a high frame rate camera. Importantly we do not need to change any tracker configuration, or change physical interfaces to devices: one just needs to write a simple display application. This allows us to monitor the latency of more complex system assemblies. The system is very simple to set up and thus we hope it makes latency measurement much more accessible in the laboratory.

3 Latency Measurement with Video

The method we propose, which we will term the *sine fitting method,* uses a tracked pendulum. We attach a small light to the pendulum and then video the pendulum and a screen behind it which shows a simulated image whose position is driven by the tracking information. We do not need to take care to calibrate the camera position, we just need to ensure that the full range of motion of the pendulum and the simulated image are on the video frame. Figure 1 shows a simple pendulum set-up that we used in the experiments in Section 4. Figure 2 shows a representation of the video taken of the pendulum and the simulated image.

The advantage over previous methods is that because we know the motion of the tracker, we can fit a curve to the discrete positions extracted from the video. By extracting position information from every video frame, we should be able to reconstruct the motion accurately and thus achieve better time discrimination than simply counting frame offsets, or sampling single positions of the pendulum motion.

The current Matlab implementation of the sine fitting method has the following stages:

1. Load video and select regions from the video that contain the physical pendulum and simulated image.
2. Subtract the background.
3. Threshold and extract the pendulum and simulated image regions.
4. Find centroids of pendulum regions and simulated image regions and extract the horizontal displacements.
5. Remove any outlier displacements, and replace them by the average of adjoining pixels.
6. Normalize displacements.

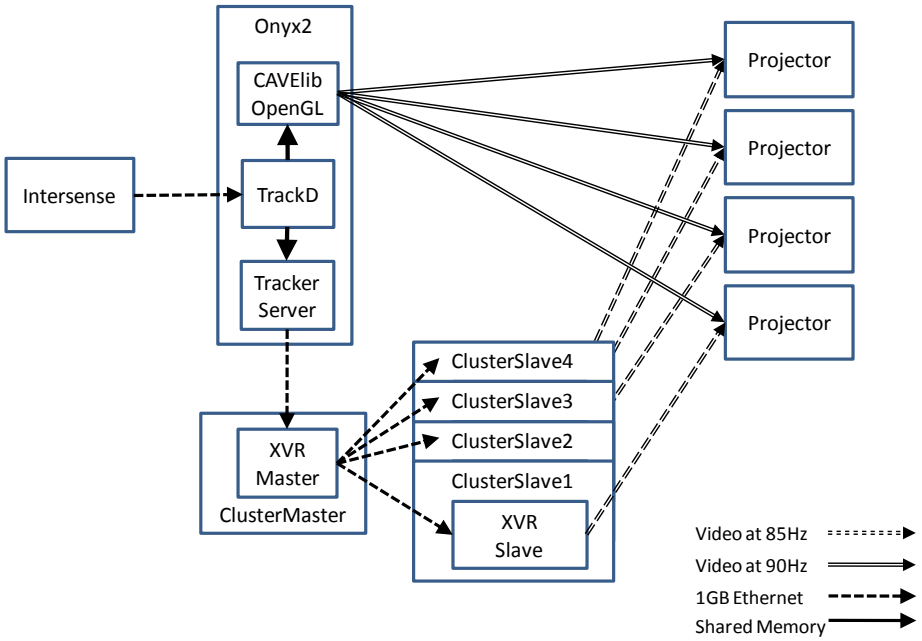

Figure 4: Outline architecture for the system being measured. There are two paths to generate images, one via CAVElib/OpenGl on the Onyx2, and one via a PC cluster.

7. Fit one sine curve to the pendulum displacements and a second to the simulated image displacements.

8. Convert the phase difference in frames in to ms.

Most of these stages are quite simple and use standard Matlab functionality. For Step 7, we have two series of displacement values normalized to [-1,1]. For each series we first estimate the dominant frequency and the phase using the technique of Quinn & Fernandes [1991]. To each series we then fit a function of the form sine(ax + b) using the built-in Matlab function lsqcurvefit. Figure 3 shows an example of the fit of the two curves to 100 frames of the video sequence represented in Figure 2.

We did a number of assessments of the potential sensitivity of this method using simulated video. Under these ideal situations, where the targets were solid red and solid green and exactly two pixels square, we were able to reconstruct the latency to within 1ms, when the latency was simulated in the hundreds of ms. We experimented with adding random noise to the displacements. On a video 200 pixels across, normally distributed noise with a standard deviation of 10 pixels was added after we extracted horizontal positions at Step 4. This degraded the accuracy of the latency estimation to +/- 5ms. This level of accuracy would still be useful as a measure, and this level of noise is significantly higher than the noise in the videos that we examined.

4 Demonstrations

We examine the sensitivity of the sine fitting method (SFM) by estimating the latencies of a typical large visualization facility. This has two different image generators with unknown end-to-end latencies. We also measure the impact of delaying the rendering by one or more frames, which will have a well known latency impact. For comparison, we make estimations using a second latency measuring method based on a frame counting method (FCM) using a high-speed video camera.

4.1 System Structure

The system layout of a CAVE™-like system is shown in Figure 4. The tracking system is a wired Intersense IS-900 with three tracking devices connected [Intersense 2008]. The device was configured with no latency compensation, high precision and high sensitivity. The device is connected over RS232 at 38.4kbaud in binary mode to an SGI Onyx2. The Onyx2 has 8 MIPS R12000 processors, 8GB RAM and four InfiniteReality2 graphics pipes. The tracking information is read by a TrackD process [TrackD 2008].

The first image generator is an OpenGL/CAVElib™ [CAVELib 2008] program that runs on the Onyx2. This program connects to the TrackD process via shared memory. It generates four mono or time-sequential stereo video signals at 90Hz. These drive CRT projectors at 1028x768 pixels. The CAVE™-like system comprises three 3m x 2.2m walls and a 3m x 3m floor.

The second image generator is a self-built PC cluster. This has a cluster master node with 2GB RAM, and dual 1.8GHz Intel processors, and four cluster slave nodes with 1GB RAM, single 2.7GHz Intel processors and GeForce Quadro 5600 graphics cards. All cluster nodes run Windows XP. Each cluster slave node generates a single mono or time-sequential stereo video signal at 1028x768 @ 85Hz. To retrieve tracking information, a custom tracker server process on the Onyx2 connects to TrackD and distributes the readings over UDP at 200Hz. On the cluster master node, we use the XVR software to create our applications [XVR 2008]. This reads the tracking information coming from the Onyx2 in a dedicated process, and consumes the UDP tracking data at 200Hz. The XVR process on the cluster master delegates rendering to the cluster slaves using a form of distributed OpenGL. The slave nodes are responsible for buffering OpenGL and rendering two views; the master is unaware that stereo rendering is occurring.

Configuration	Frame Delay	Samples	SFM Mean (s)	SFM Std. Dev.(s)	FCM Mean (s)	FCM Std. Dev (s)
Onyx2, Mono	0	12	0.090	0.0076	0.104	0.0222
	1	4	0.111	0.0025	0.122	0.0159
	2	4	0.133	0.0068	0.123	0.0233
	3	4	0.154	0.0113	0.136	0.0163
	4	4	0.174	0.0037	0.177	0.0166
	5	4	0.197	0.0058	0.192	0.0370
Cluster, Mono	0	12	0.064	0.0096	0.072	0.0135
	1	4	0.078	0.0016	0.106	0.0170
	2	4	0.096	0.0066	0.116	0.0196
	3	4	0.116	0.0093	0.142	0.0117
	4	4	0.131	0.0029	0.131	0.0194
	5	4	0.150	0.0123	0.168	0.0306
Onyx, Stereo Left Eye	N/A	8	0.085	0.0076	0.104	0.0208
Onyx, Stereo Right Eye	N/A	8	0.096	0.0092	0.107	0.0234
Oynx, Mono, Offset mark	N/A	12	0.091	0.0074	0.105	0.0160

Table 1: Estimated latencies with the sine fitting method (SFM) and frame counting method (FCM) for several system configurations.

The networking between Onyx2 and cluster master is an un-congested 1GB Ethernet. The cluster master node has a second 1GB Ethernet interface to the cluster slave nodes.

For both applications, a single green quadrilateral was drawn. Only the tracker position was used, so the polygon did not rotate on the screen. This is what we refer to as the simulated image. Both applications allow the displays to be configured to render in time-sequential stereo or mono. In both applications we added functionality to optionally buffer and delay input tracker readings by between 1 and 5 frames.

4.2 Implementation of Sine-Fitting Method

We used a Panasonic NV-GS300 Camera which is a 3 CCD camera, recording at PAL frame rates (25Hz, 50Hz interlaced). We used a 3CCD camera because we track two different colors on the screen. A slightly different method using the same color for the LED and the simulated image would mean that any camera could be used without worrying about timing differences due to the camera's color processing. The video captured to a laptop was 720 by 576 in size. We de-interlaced the video by removing the second field, and down-sized the video so that it was 320x240 @ 25 Hz. One could extract both fields, use a progressive scan camera, or a camera with NTSC rates to get a higher frame rate video. For our tests we would either ensure that the display was in mono, or the video was shot through stereo glasses so that there was not a double image of the simulated image on the display.

We hung the tracker from the ceiling of the display area so that the swing had radius of approximately 0.7m. To ensure a vertical swing, we suspended both ends of the tracker which is a pistol-grip shaped device. The assembly is shown in Figure 1. Once the pendulum had swung a few times, any horizontal rotation was imperceptible to the naked eye leaving only motion in a vertical plane. We attached a red LED to the tracker. We used a green quad as the simulated image, because green was the brightest tube in our CRT projectors. The lowest point on the swing of the green quadrilateral was about 0.7m below the top of the front screen. The latency to each screen should be the same because the renderings are frame-locked together and the video generation uses genlocked hardware.

We found through informal experimentation, that about 4 seconds of video, and 3-4 swing cycles was sufficient to get a good estimate of latency. The filtering and numerical estimation parts of the method are fast enough that in principle the latency estimation could be done in real-time. In the results presented below, the images of the pendulum and simulated image were about 3x3 pixels, and would move between ¼ and ½ of the way across the screen.

4.3 Implementation of Frame Counting Method

We gathered comparative latency estimates using a Fastec TroubleShooter camera. This was configured to capture video at 640x480 pixels at 500Hz. Latency was estimated by counting the number of frames between the pendulum and the simulated image reaching extreme positions. This process was done by hand using image enhancement software because of low light capture by the camera at 500Hz. It is easy to pin-point the frame where the physical pendulum reaches its extreme position. It is harder to do the same for the simulated image. Firstly, because the screen is a CRT running at 85Hz or 90Hz, only every 5^{th} or 6^{th} frame on the video captured by the Fastec camera shows the CRT illuminated because of the CRT's scanning. The simulated image may also be static on for several frames at the extreme because of small sub-pixel movements. Thus the frame number for the extreme of the simulated image was chosen to be middle of the frames where the simulated image appeared to be at its extreme.

5 Results

A number of different measurements were made on the system. A summary of the findings is presented in Table 1. Both latency estimation methods estimate gave the Onyx2 a higher latency than the PC cluster system, with estimates of 90ms for the SFM and 104ms for the FCM for the latency of the Onyx2, compared to 64ms by the SFM and 72ms by the FCM for the cluster. However, we note that the SFM gives a much lower standard deviation in its estimation compared to the frame counting method. The difference between the estimates of the two system latencies is notable because despite the Onyx2 hosting the physical interface to the tracker, the superior clock-rates of the components of PC cluster system means that it has a lower end-to-end latency. We

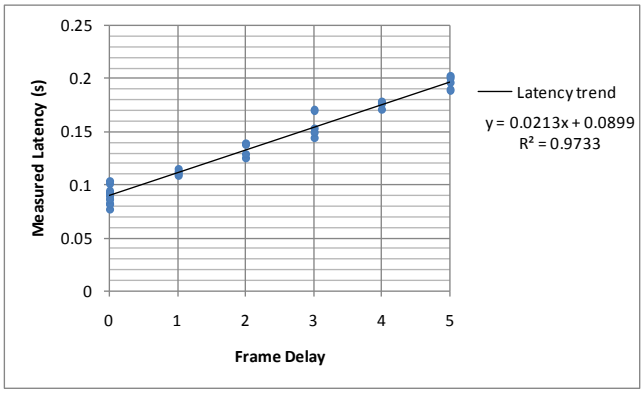

Figure 5: The estimated latency of the Onyx 2 system at different frame delays with the sine-fitting method

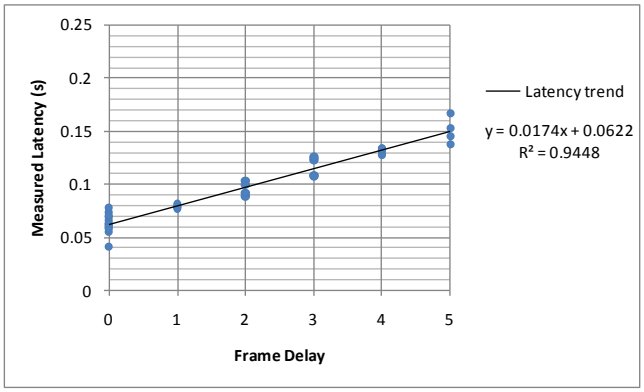

Figure 6: The estimated latency of the PC cluster system at different frame delays with the sine-fitting method

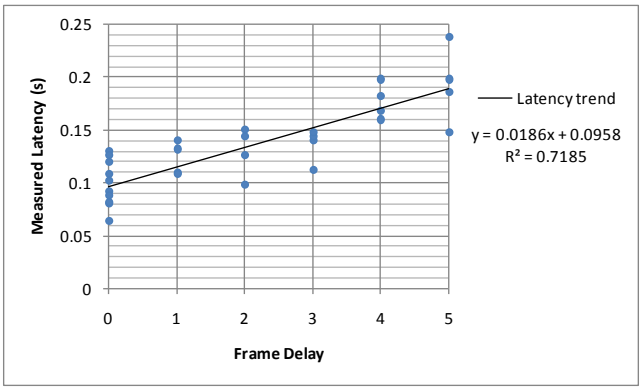

Figure 7: The estimated latency of the Onyx 2 system at different frame delays with the frame-counting method

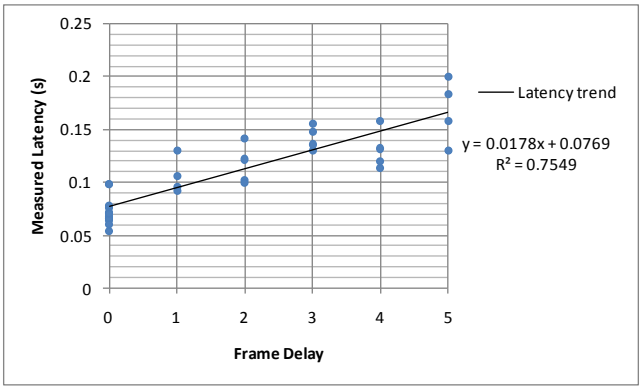

Figure 8: The estimated latency of the PC cluster system at different frame delays with the frame-counting method

note that for the SFM the standard deviation of the readings for Onyx2 is slightly lower than that for the PC cluster. Extra jitter would be expected for the PC cluster because of the additional processes involved.

We can also investigate the effect of frame delay on each system. The results are also given in Table 1 and plotted in Figure 5 and Figure 6 for the SFM and Figure 7 and Figure 8 for the FCM. We have fitted a trend line to each plot. The trend lines in Figure 5 and Figure 6 confirm the base latencies with no added frame delay (63ms and 90 ms) that were reported for the SFM. The slopes are different with each frame delay on the PC cluster causing an extra 17ms latency and each frame delay on the Onyx2 causing an extra 21ms latency. These indicate, respectively, frames at 60Hz and 47Hz. The former frequency is precisely the frequency of update of the monitor of the cluster master: this indicates that the XVR system is bound by the refresh rate of the cluster master's display (60Hz), rather than the cluster slaves' displays (85Hz). The rate of 47Hz, is fractionally higher than half the video output rate of the Onyx2 (90Hz). This indicates that, in mono mode, the application system does not exploit the second potential frame that is available. The fit for the two trend lines is very good for the SFM.

The FCM does not give such good discrimination of the impact of latency. Although the estimated trend in latency for the cluster is calculated at 17ms, the trend line itself is a relatively poor fit to the data and the trend in latency for the Onyx2 is 18ms a frame, or 55Hz, which is not close to the expected per frame delay of 22ms (45Hz). We suggest that the poor performance of the FCM compared to the SFM is because the latter is using many more sample points, as it uses every frame of the video in its estimation.

An additional test we performed was to track the latency between the two different stereo frames for the Onyx2. For the SFM this should be quite challenging, because the theoretical latency difference is 11ms (90Hz) when the frame length of the video is 25 ms (50Hz). However, as can be seen in Table 1, the stereo left image has a latency of 85ms, and the stereo right 96ms. Although the standard deviations of the two are relatively high compared to the latency difference, we performed a Student's unpaired, two-tailed t-test to compare the two distributions of estimates. The t value was 2.619, and with 14 degrees of freedom, this gives a significant difference at $p = 0.0202$ (i.e confidence above 95% to reject the null hypothesis that the two distributions are the same). There was no significant difference in the frame times between stereo left eye and right eye for the FCM.

A final, inconclusive test that was performed was to attempt to track the latency difference between two different areas of the screen. This result is the last presented in Table 1. We offset the simulated image so that it was 0.66m lower on the screen. With 2.2m high screens, this is 3/10 of the screen further down. Thus if the screen refreshes at 90Hz, with a CRT this should be approximately 3ms additional latency. Although the SFM does estimate a difference in mean latencies, the difference is only 1.3ms, and the standard deviations indicate that the difference is

not statistically significant (t value of 0.4236 and 22 degrees of freedom, give p=0.67). The FCM does not detect a significant difference either.

6 Conclusions

We have demonstrated a new, simple to set up and relatively sensitive latency measurement method. The sine fitting latency estimation method is easy to configure as it simply needs a tracker to be hung from a pendulum and it only requires a standard video camera. This also makes it relatively flexible to install: there is no need to bring additional equipment in to the tracked space and the video camera can be fixed a convenient distance away. A simple display application needs to be written or an existing application needs to be modified.

For more general applications, where the developer wants to include the latency of their own application in the measurement, there are several strategies that could be used to generate the required visual feedback without including incurring extra latency. The rendering required is only a few pixels, enough to be tracked on the video, so does not intrinsically take much time. If there is sufficient rendering time, the frame buffer can be cleared after the full render, and the simulated image drawn. To avoid a frame buffer clear, the normal application rendering could be done to an off-screen buffer, to a different viewport, or in certain circumstances the rendering pipeline or shader configured to discard RGB pixels after the z-test (thus incurring the full cost of the shading), or with a color mask. After any alterations to the rendering, the developer needs to check that the frame rate does not change; this should imply that the rendering latency hasn't been increased.

With the sine fitting method we were able to estimate end-to-end system latencies of two different image generators for a CAVE™-like system. We were able to accurately detect the expected impact of an introduced frame delay in the application system. Further we were able to reliably detect the timing offset between the two eyes in a frame sequential stereo system, a difference of 11ms. With the data we gathered we were not able to reliably detect a timing difference of 3ms. However it is worth reflecting that at this temporal resolution, we might be frustrated either because our latency detection process is not sensitive, or because 3ms is within the normal variation in the end-to-end latency of the system. Based on the discussion of Section 2.1, for the system we investigated, we can predict an uncertainty of ~5ms in the end-to-end system latency, because the trackers, rendering and displays all operate at different frequencies. Thus we suggest that our latency measurement process is sufficient to detect the types of difference that might be important in normal laboratory conditions.

The full Matlab code, the test applications written in CAVElib and XVR and an example video can be found on the author's home page: http://www.cs.ucl.ac.uk/staff/A.Steed

Acknowledgements

Many thanks to Dr. Alan Wilson from the Royal Veterinary College, London for the loan of the Fastec TroubleShooter camera.

References

ADELSTEIN, B., JOHNSTON, E. and ELLIS, S. 1996. Dynamic response of electromagnetic spatial displacement trackers. Presence: Telepresence and Virtual Environments, 5(3):302-318.

BRYSON,S., and FISHER. S. S. 1990. Defining, Modeling, and Measuring System Lag in Virtual Environments. Stereoscopic Displays and Applications I, Proceedings SPIE 1256, 98-109.

CAVElib™, VRCO, 2008. http://www.vrco.com/

CRAIG, S. J., REID, L., and KRUK, R. 2000. The effect of visual system time delay on helicopter control. Proceedings of the IEA 2000/HFES 2000 Congress, 3-69-3-72.

CRUZ-NEIRA, C., SANDIN, D.J. and DEFANTI, T.A. 1993. Surround-screen projection based virtual reality: the design and implementation of the CAVE. Proceedings of the 20th annual conference on Computer graphics (SIGGRAPH '93), 135-142.

ELLIS, S.R., ADELSTEIN, B.D., BAUMELER, S. , JENSE, G.J. and JACOBY, R.H. 1999. Sensor Spatial Distortion, Visual Latency, and Update Rate Effects on 3D Tracking in Virtual Environment. Proceedings IEEE Virtual Reality (IEEE VR'99) Conference, 218-221.

ELLIS, S. R., WOLFRAM, A., AND ADELSTEIN, B.D. 2002. Large amplitude three-dimensional tracking in augmented environments: a human performance trade-off between system latency and update rate. Proceedings of HFES. pp. 2149-2154.

HE, D., LIU, F., PAPE, D., DAWE, G. and SANDIN. D. 2000. Video-Based Measurement of System Latency, International Immersive Projection Technology Workshop, Ames IA, USA.

INTERSENSE IS-900, 2008. http://www.isense.com/

JACOBY, R. H., ADELSTEIN, B. D., and ELLIS, S. R. 1996. Improved temporal response in virtual environments through system hardware and software reorganization. Proceedings of the SPIE 2653, Stereoscopic displays and virtual reality systems III, 271-284

LIANG, J., SHAW, C., and GREEN, M. 1991. On Temporal-Spatial Realism in the Virtual Reality Environment. Proceedings of the fourth annual symposium on user interface software and technology 19-25.

MEEHAN, M., RAZZAQUE, S., WHITTON, M. C., and BROOKS, F. P. 2003. Effect of Latency on Presence in Stressful Virtual Environments. In Proceedings of the IEEE Virtual Reality 2003 (March 22 - 26, 2003). VR. IEEE Computer Society, Washington, DC, 141-138.

MINE, M. 1993. Characterization of end-to-end delays in head-mounted display systems. Technical Report TR93-001. Department of Computer Science, University of North Carolina at Chapel Hill.

MILLER, D. and BISHOP, G. 2002. Latency Meter: a Device for Easily Monitoring VE Delay, in Proceedings of SPIE Vol. #4660 Stereoscopic Displays and Virtual Reality Systems IX, San Jose, CA.

QUINN, B.G. and FERNANDES, J.M. 1991. A fast technique for the estimation of frequency, Biometrika,. 78(3), 489-497

SWINDELLS, C., DILL, J. C., and BOOTH, K. S. 2000. System lag tests for augmented and virtual environments. In Proceedings of the 13th Annual ACM Symposium on User interface Software and Technology (UIST '00). ACM, New York, NY, 161-170.

TRACKD, VRCO. 2008. http://www.vrco.com/trackd/Overviewtrackd.html

XVR. 2008. VR media, http://www.vrmedia.it/