Latency Compensation for Head-Mounted Virtual Reality

Jason Jerald

Department of Computer Science
University of North Carolina at Chapel Hill

jjerald@cs.unc.edu

ABSTRACT

Latency greatly degrades the usability of head-mounted virtual reality systems. This paper discusses consequences of latency, examines sources of latency, investigates techniques to reduce latency, and proposes to integrate existing solutions into a system that approximates zero latency for the most critical types of errors. First, effects of latency are examined such as degraded user experience, lowered user performance, and simulator sickness. System delay is then analyzed from the tracker to the display. Advantages and limitations of existing methods for reducing latency are discussed, followed by an explanation of how software performance engineering allows one to analyze and optimize performance. Finally, a proposed system is presented that integrates three techniques: system delay reduction, head-orientation prediction, and post-rendering image correction in hardware. The proposed system could approximate zero latency for head-mounted virtual reality.

Keywords

Virtual reality, system delay, latency, human-computer interaction, software engineering, systems analysis and optimization, system and hardware architecture, prediction

1 INTRODUCTION

The fundamental idea of *virtual reality* (VR) is to present the user with perspective images that change appropriately at correct times as she moves. A *head-mounted display* (HMD) is a tracked display rigidly attached to a user's head that presents the visual representation of the world [Sutherland 1968]. *System delay* is defined as the true end-to-end delay of the entire system from the time of tracking to the time of display. If a VR system does not compensate for this system delay then the images are not presented at the correct times. *Latency* is defined as the effective delay of the system as perceived by the user. Without prediction or correction, latency is equal to system delay. If one could accurately predict and correct for system delay so that the image looks correct for the time of display, then the effective latency would be equal to zero.

Latency is a primary factor in detracting from the sense of presence in VR [Meehan *et al.* 2003]. The slower the response of the system, the more the visual world seems to lag behind the user's actions. Reducing latency (improving responsiveness) provides more believable virtual worlds and is important for natural and precise interaction within these environments. However, exact latency requirements of VR are not yet known. A tool is needed that would allow researchers to determine the threshold of human perception of latency and the point when latency begins to impact task performance. A system with near-zero latency would allow one to slowly raise latency levels until human subjects are able to differentiate among the conditions.

This paper discusses latency and its effects on users of head-mounted VR systems. Various topics are investigated in order to develop a better understanding of latency causes, effects, sources, and reduction. Topics include human computer interaction, software engineering, and hardware architecture. By applying concepts from these fields, I describe how to reduce latency and, consequently, improve VR systems. Methods of system analysis and optimization, prediction, and post-rendering image correction are applied to achieve the goal of near-zero (within a millisecond) latency for the most important type of errors.

Section 2, Degrading Effects of Latency, explains why users of tracked HMDs are susceptible to the ill effects of latency. Problems include dynamic error, oscillopsia, degraded visual acuity, and degraded human performance. The most critical type of latency-induced error — error due to head rotation — is also discussed.

Section 3, Sources of System Delay, investigates latencies associated with various hardware and software components in a typical VR system. Tracking, networking, application, rendering, display, and synchronization delays all contribute to total system delay.

Sections 4–6 discuss three methods of reducing latency — system analysis and optimization, prediction, and post-rendering image correction. Section 4, System Analysis and Optimization, describes how to reduce system delay by pinpointing delay components. Software performance engineering, measurement, and timing diagrams help to better understand system delay. The section ends with an example that reduces system delay by reducing synchronization delays. Section 5, Prediction, explains how to predict head movement. Prediction is required because instantaneous computation is impossible – there will always be some system delay. By the time a new image is computed and presented, the user may be looking in a different direction. Thus, head pose needs to be predicted for the time the new image will be displayed to the user. Section 6, Post-Rendering Image Correction, discusses how post-rendering image correction can correct for a small head-rotation error. The system determines the amount of error and corrects for it by modifying the image scan-out at the last possible moment before display.

Section 7, Proposed Method, proposes to integrate the above three methods of reducing latency. The method consists of minimizing system delay and delay variance, estimating the average system delay (the prediction interval), predicting user orientation for the instant when the image appears, rendering for the predicted orientation, and correcting the remaining yaw orientation (i.e. looking left/right) error at the last possible moment. The end result is a system where users perceive near-zero latency. Figure 1 (top) shows a simplified pipeline of a typical VR system. The proposed system, shown in Figure 1 (bottom), integrates a *predictor* and *corrector*. Rate gyroscopes give angular velocity and are used as input to the predictor and corrector. The new head pose that is output from the predictor is the best guess as to where the user will be looking when the image is displayed. Just before display of each scanline, the corrector computes the current head yaw error and applies a correction.

The paper ends with Section 8, Conclusion and Future Work.

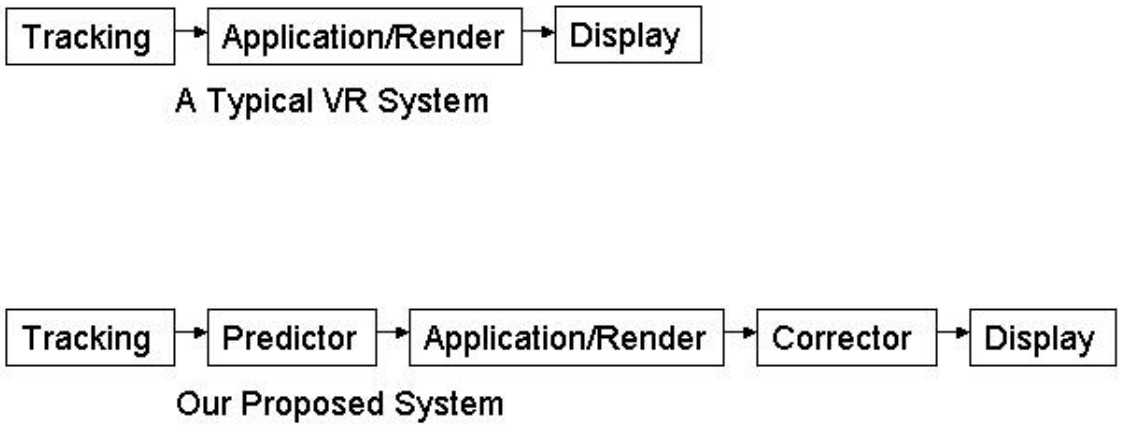

Figure 1. Overview of a typical VR system and the proposed system.

2 DEGRADING EFFECTS OF LATENCY

2.1 Registration Error

Registration error is the difference between where a pixel appears to be and where the pixel would appear if it were a physical object in the real world [Holloway 1995]. Registration errors are more obvious with optical-see-through displays than non-see-through displays, since users can directly compare the real world with the virtual world. These errors can further be classified as *static* or *dynamic* [Azuma and Bishop 1994]. *Static errors* result from optical distortion, incorrect viewing parameters, imprecise calibration of equipment, etc., and occur even when the user keeps her head completely still. *Dynamic errors* result from temporal mismatch between head movements and the visual display; they do not occur until the user moves her head.

Latency causes dynamic errors when there is head movement. As latency and head movement each increase, dynamic errors increase. This causes the graphics to appear to "swim" or chase after the real world as the user moves her head. Even for moderate head velocities, latency causes more registration error than all other registration errors combined [Holloway 1995]. For most virtual environments, yaw error is larger than other errors as a result of the user turning the head left/right both faster and more often than other movements.

Dynamic error can be computed by simple mathematics. For example, given a display width of 640 pixels and a field of view of 32 degrees, there are 0.05 degrees per pixel. Assuming a moderate yaw head rotation of 50 degrees per second, the horizontal error is

(50 degrees / second) / (0.05 degrees / pixel) = 1 pixel / ms.

This example illustrates one pixel of error for every millisecond of latency. Faster head rotations have more pixel error. This calculation shows latency must be reduced to the millisecond range to obtain pixel accuracy. Since today's best VR systems have latency in the 30 millisecond range, this simple requirements analysis suggests there is much room for improvement.

2.2 Usability Problems

Error due to latency has serious usability consequences. Latency causes visuals to lag behind other perceptual cues creating sensory conflict. Contradiction among cues can result in oscillopsia, degraded visual acuity, and degraded human performance. In this section, information is derived from [Allison *et al.* 2001] unless otherwise referenced.

Oscillopsia

Oscillopsia is the perceived oscillation of the entire visual world when in reality the world is stable. A person with oscillopsia perceives the visual world to swim or oscillate in space and loses a sense of perceptual stability of the environment. This symptom has been reported with drug toxicity, brain injury, and damage to the vestibular system.

Oscillopsia can result from the mismatch between head motion, eye movement, and visuals. As the head moves, the motion is detected by the vestibular system in the inner ears, giving the perception of head motion even if there is no visual change. Vestibular signals cause compensatory eye movements to keep eye gaze stable. The visual system assumes the world is stable even with eye or head movements. Optic flow is movement of the entire visual image; vection is the perception of self-motion generated by optic flow. If vection does not match eye gaze change and head movement as determined by the vestibular system, then oscillopsia may result.

Latency in VR can cause oscillopsia, which increases with increasing latency and increasing head movement. VR should not cause these oscillopsiac effects, and the world should appear stable to a user as she moves within that environment.

Degraded Visual Acuity

In addition to Oscillopsia, latency can also cause degraded vision. Images moving faster than two degrees per second on the retina result in motion blur as perceived by the user. As the user moves her head then stops, the image is still moving when she has stopped (assuming some latency). Motion blur and degraded visual acuity result.

Degraded Human Performance

The level of latency necessary to impact performance negatively may be less than the level of latency that can be perceived. In non-see-through applications, although users may not consciously notice latency, performance may suffer.

[So and Griffin 1995] study the relationship between latency and operator learning. The task consisted of tracking a target with the head. Training did not improve performance when there was 120 ms of latency or more. Thus, the subjects were unable to learn to compensate for these latencies in this task.

Discrimination of Inconsistent Latency

[Ellis *et al.* 1999] have shown that users are just as sensitive to changes in latency with a small base latency as those with a larger base latency. Users are able to reliably discriminate increases of 33 ms independent of base latency. These results suggest consistent latency is an important factor to consider in VR.

2.3 Determining Acceptable Latency

Exact latency requirements for head-mounted VR are not yet determined due to the limitations of current technology. Systems have been built with latencies in the 30 millisecond range but it is unknown if latencies below this range significantly effect the user. One motivation for building a low-latency VR system is to perform user studies to discover the threshold of human perception of latency. Acceptable latency cannot be determined until such a system exists.

3 SOURCES OF SYSTEM DELAY

[Miné 1993] and [Olano *et al.* 1995] characterize system delays in VR and discuss various methods of reducing latency. This section expands on these concepts of breaking down system delay. System delay is the sum of delays from tracking, networking, application, rendering, and synchronization between components. Figure 2 illustrates components contributing to system delay in a typical VR system.

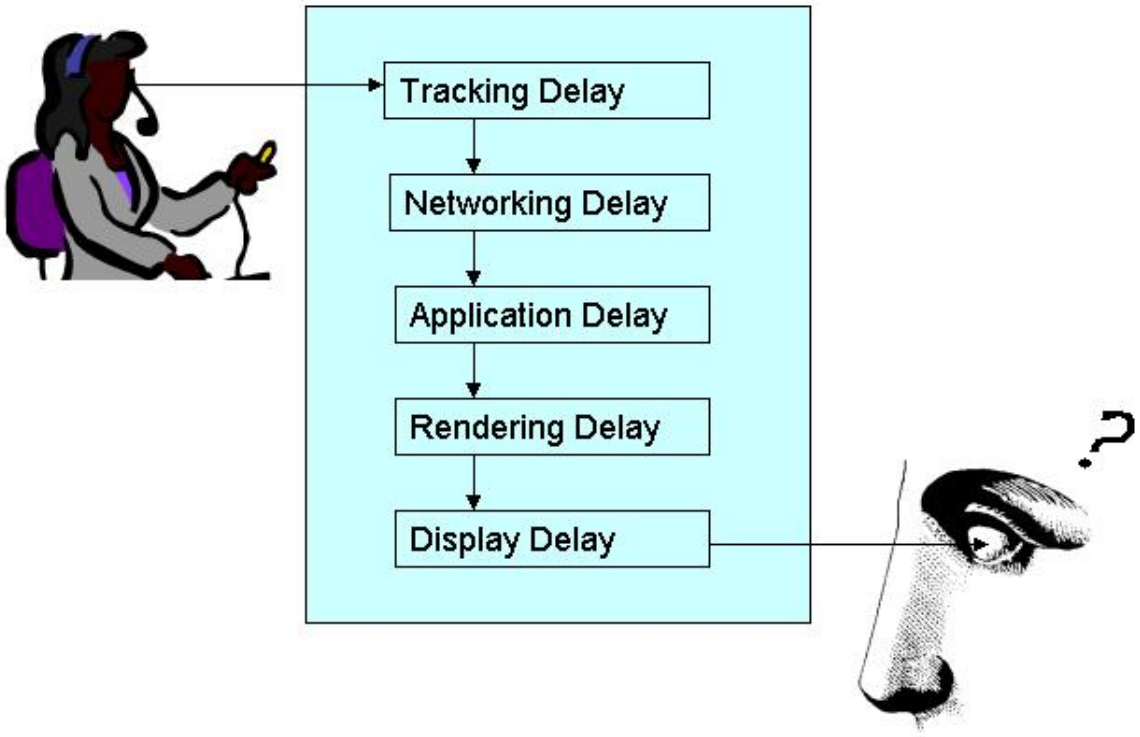

Figure 2. Sources of system delay in a typical VR system.

3.1 Tracking

Tracking technology can include techniques that complicate latency analysis. For example, many tracking systems add filters to smooth jitter. If filters are used, the resulting output pose is only partially determined by the most recent tracker reading. Some trackers use different filtering models for different situations — latency during some movements may differ from that during other movements. For example, the 3rdTech HiBall tracking system [3rdTech 2002] allows the option of using multi-modal filtering. A low-pass filter is used to reduce jitter if there is little movement, whereas a different model is used for higher velocities.

3.2 Networking

Communication between disparate VR system components often uses a standard network. For example, the tracking component is often located on a machine separate from the rendering computer. Network delay can vary due to protocol stacks, interrupts, send rates, buffering, etc.

Communication between remote components is normally implemented using TCP/IP, since easy-to-use libraries exist. Using TCP/IP may require a private network so that unrelated network traffic will not cause unexpected delays. If sub-millisecond clock synchronization between components is required, better protocols may be necessary.

Networking tasks incur operating system overhead and can contribute to system delay. Whereas delay is reduced by having the latest information from preceding system components as soon as that information is available, many messages may overload the system. A high rate of context switches (e.g., 1000 interrupts per second) can dramatically reduce system performance. In such a case, the system may spend a large proportion of resources switching between tasks, resulting in increased system delay.

3.3 Application

Application delay is task-dependent and also varies among virtual environment models. The application component often consists of application-specific computation that is not tightly coupled with tracking and rendering and therefore may be able to be executed asynchronously from the rest of the system. This is discussed further in Section 4.4, Reducing Synchronization Delays.

3.4 Rendering

Like application delay, rendering delay depends on the complexity of the environment. The *rendering rate* is the number of times the system can render the entire scene per second. The *rendering time* is the inverse of the rendering rate (seconds per render) and is equivalent to rendering delay in non-pipelined rendering systems. Some systems reduce the rendering time by culling out objects outside the field of view, but at the cost of inconsistent delay. If consistent delay is important, then such culling should be turned off.

Rendering is normally computed separately from the application, typically on graphics hardware. Current graphics cards can render at rates over one hundred renders per second for hundreds of thousands of polygons per render. Although rendering is no longer a major component in most systems, it still contributes to system delay.

3.5 Display

Due to the large and difficult-to-predict response times of typical licquid crystal displays (LCDs) [Nakanishi *et al.* 2001], this paper focuses on the more consistent cathode ray tube (CRT) displays.

A CRT sweeps the display, scanning out left-to-right in a series of horizontal lines from top to bottom [Whitton 1984]. This pattern is called a *raster*, and each horizontal line is called a *scanline*. The timings of the scanlines are precisely controlled to draw pixels from memory to the correct locations on the screen. Not all pixels are drawn to the screen in a CRT at the same time. The bottom-right pixel is drawn nearly a frame time after the top-left pixel.

A *frame* is the full-resolution image that is scanned out to the display hardware. The *frame rate* is the number of frames updated to the display per second. Note this is different than the rendering rate described above in Section 3.4, Rendering. The *frame time* is the inverse of the frame rate (seconds per frame). Displays commonly have frame rates of 60 frames per second (Hz), which is equivalent to a frame time of 16.7 milliseconds. All examples in this paper use frame rates of 60 Hz.

Dual access to image memories can be accomplished by using a double-buffer scheme. The display processor renders to one buffer while the refresh controller feeds data from the other buffer to the display. The *vertical sync* is a signal that occurs at the start of each frame. Normally, the system waits for this vertical sync to swap buffers. The newly rendered image is then scanned out to the display while a yet newer image is rendered.

Waiting for vertical sync comes at the price of a large amount of variable delay, since the system must wait between 0 and 16.7 ms. If the system does not wait to swap on vertical sync, then *tearing* occurs during head movements and appears as a discontinuous image due to regions of the frame being rendered for different head poses. This originates from the buffer swap occurring while the frame is being scanned out to the display hardware. Part of the frame is from the previous rendering and part from the current rendering, hence from multiple points of view if there is any head movement. When the system waits to swap buffers on vertical sync, no tearing is evident because there is a single rendering for the entire frame. Thus, VR systems normally avoid tearing at the cost of additional and variable delay.

Tearing between two poses decreases with increasing pose coherence. As the sampling rate increases in time, adjacent pose coherence increases, resulting in less tearing. If the system were to render each pixel with the correct up-to-date viewpoint, then tearing would occur between pixels. The difference in poses between pixels would be small compared to the pixel sizes, resulting in a smooth image – without perceptual tearing. [Mińe and Bishop 1993] call this *just-in-time pixels.*

Rendering could conceivably perform at a rate fast enough that buffers would be swapped for every pixel. Although the entire image would be rendered, only a single pixel would be displayed for each rendered image. However, a 640x480 image would require a rendering rate of over 18 MHz — clearly impossible for the foreseeable future using standard rendering algorithms and commodity hardware. If a new image were rendered for every scanline, then a 640x480 image would require nearly 29 KHz – still not possible with today's hardware. In practice, today's systems can render at rates up to 600 Hz, which make it possible to show ten new images per frame time. As the rendering rate increases, the tearing becomes less evident and the system approaches a just-in-time pixels implementation.

Figure 3 shows a frame that does not wait on vertical sync superimposed over a frame that does wait on vertical sync. The figure shows what a static virtual block would look like when a user is turning her head from right to left. The tearing is obvious when not waiting on vertical sync, due to four renderings for the frame at four different times.

In current VR systems, the irregular delays created by waiting on vertical sync can be the largest source of variance in system delay. Ignoring vertical sync greatly reduces overall latency and its variability (at the cost of image tearing).

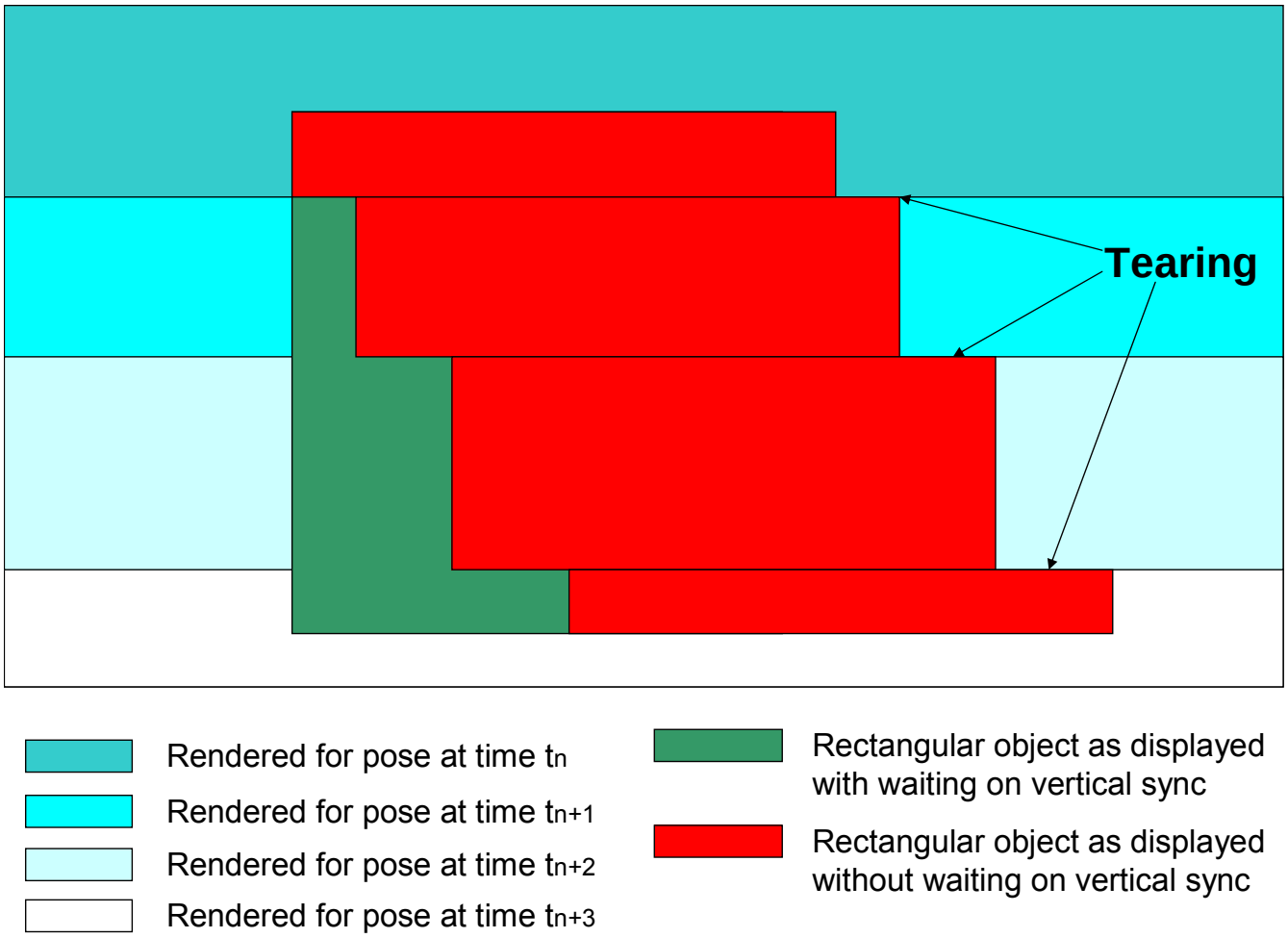

Figure 3. A frame showing a rectangular object as the user is looking from right to left with and without waiting to swap on vertical sync. The result of not waiting to swap vertical sync to swap is perceived as image tearing.

3.6 Synchronization

Total system delay is not simply a sum of component delays. Synchronization among components contributes to system delay and delay variability even if individual component times are known.

Pipelines stages depend upon data from the previous stage. When a stage starts a new computation and the previous stage has no updated data, then old data must be used. Although the previous stage may almost be complete, the system does not know this, and data that are nearly an entire stage time old are used.

Trackers provide a good example of a synchronization problem. Commercial tracker vendors report the latency of their systems. However, this is normally the response time or minimum delay incurred when the tracker outputs information at the same time the next stage of the pipeline requires that information. The tracking update rate (outputs per second) is also a crucial factor and affects both average delay and delay consistency. The system can require the latest pose at any time and the resulting delay may be up to a full tracker report time more than the response time. If a tracker reports 50 times per second then the average delay = response time + (1 / 2) * (1 / 50) = response time + 10 ms. The delay range = average delay +/- (1 / 2) * (1 / 50) = response time + 10 +/- 10 ms. A high update rate is essential for low and consistent delay.

4 SYSTEM ANALYSIS AND OPTIMIZATION

Software performance engineering can be used to reduce system delay to levels that meet requirements. Delay measurements and timing diagrams help to develop a better understanding of system delay. This section ends with an example of how to reduce system delay using these concepts.

4.1 Software Performance Engineering

Software performance engineering (SPE) is a method of constructing software systems to meet responsiveness goals [Smith 1986]. SPE models software requirements and designs. SPE also evaluates whether predicted performance metrics meet the specified goals. Performance refers to response time or throughput as perceived by users and is the primary concern. If performance goals are not met, then developers propose and assess alternatives. The process of detailed design, coding, and testing continues in order to develop more precise models and predicted performance until requirements are met.

Figure 4 shows the core SPE methodology. Each iteration of this figure is a life-cycle phase. Systems go through various life-cycle phases from creation to end of maintenance. A key part of the SPE method is collecting data to determine if the current implementation meets requirements and if further optimization is needed. If modifying the current life-cycle phase cannot meet requirements then the system must be reconfigured and a new life-cycle phase is started.

Figure 5 shows the SPE method applied to the proposed system. The developer first configures the system and models the virtual world. The system is then tested while gathering timings of total system delay as well as various component timings. If the system delay is not acceptable, the developer must determine the bottlenecks and decide if they can be removed. If the bottlenecks can be removed, one does so and goes back to the gather-timings stage. If little gain can be had from optimizations then the system must be simplified in some way. This simplification may include reducing the geometric complexity of the virtual world, simplifying the physics rules of the simulated world, etc. The developer then iterates though new life-cycle phases until the system delay is reduced to the desired level. If the system is changed in a significant way that results in a delay above an acceptable level (e.g. a more detailed model is substituted into the world), then the developer must again iterate through a new life-cycle phase.

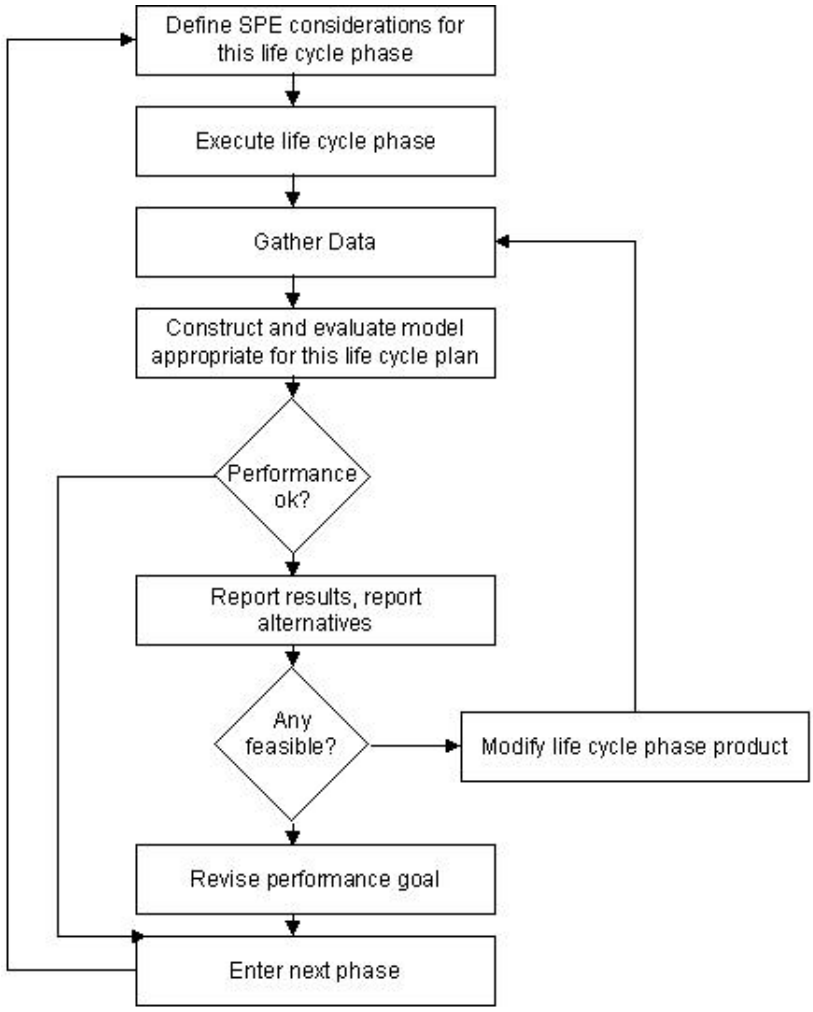

Figure 4. SPE Methodology Diagram.

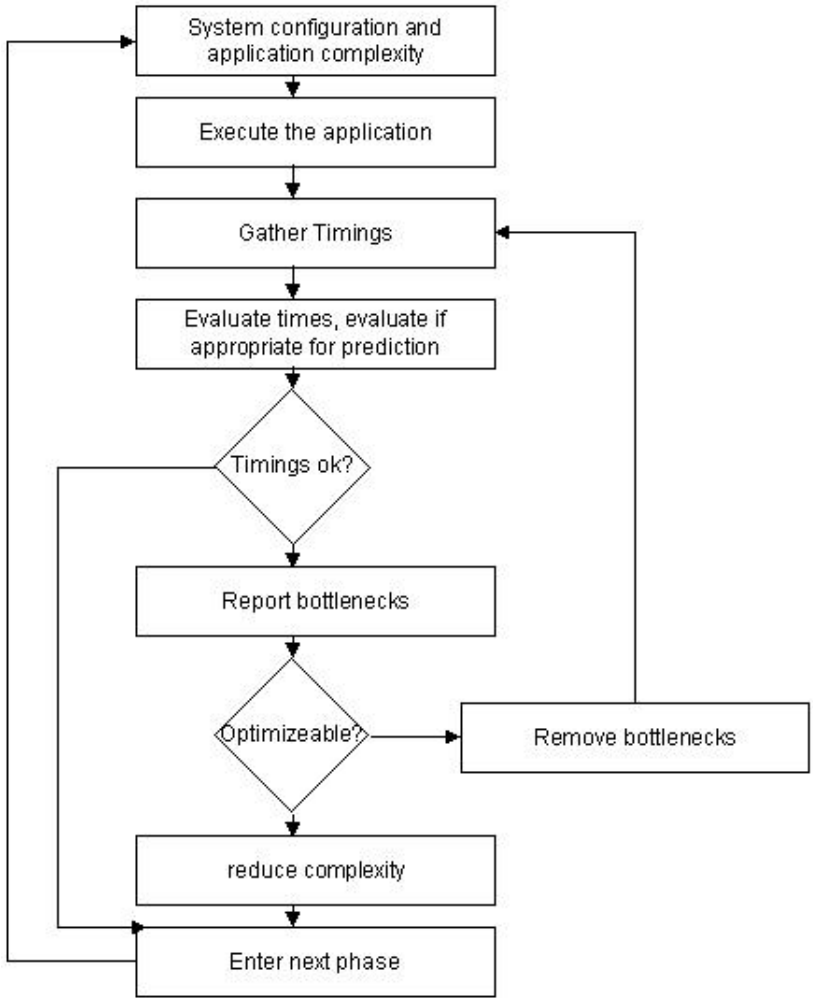

Figure 5. SPE methodology diagram applied to reducing latency.

4.2 Measurement

To better understand system delay, one must measure timings for not only the end-to-end system delay but also for sub-components of the system. Statistical measures such as the mean and standard deviation can be derived from several such measurements.

The latency meter [Miller and Bishop 2002; Miné 1993] is a device that measures system delay. This device sends a signal to the oscilloscope as the arm of the latency meter crosses the vertical (where the vertical is defined as the low point of the arc of a pendulum's motion). The latency meter test application then renders an alternating white/black screen when the application senses that the tracker crosses the vertical. A photodiode attached to the display then sends another signal to the oscilloscope when it senses the change of white to/from black. The difference of the times between the two signals, as measured on the oscilloscope, is the system delay. The video signal can also be sent directly to the oscilloscope. Since the color change is white to/from black, one can look at any of the rgb video signals to see when the change occurs. In this case the delay measured is total system delay minus the display delay.

Timings can further be analyzed by sampling signals at various stages of the pipeline and measuring the time differences. The parallel port can be used to output timing signals from both the tracking PC and application PC. These signals are precise in time since there is no additional delay due to a protocol stack; writing to the parallel port is equivalent to writing to memory;.

Synchronization delays between two adjacent stages of the pipeline can be determined indirectly through measurement. If the delays of individual stages are known, then the sum of two adjacent stages can be compared with the measured delay across both stages. The difference is the synchronization delay between the two stages.

4.3 Timing Analysis

This section presents an example timing analysis and discusses complexities encountered when analyzing system delay. Figure 6 shows a timing diagram for a typical VR system. The display stage is considered to compute discrete frames for this analysis even though individual pixels are scanned out at different times. The display stage always begins computation (scanout) at the time of vertical sync

because the system waits to swap buffers then. The image delays are the time from the beginning of tracking until the start of the display of a frame (i.e., they are equivalent to system delay minus display delay).

As can be seen in the figure, the rendering stage cannot start computing a new result until the application stage first completes and then the rendering stage completes its current computation. The other stages cannot start a new computation until similar requirements are met. In several locations, a stage computes a redundant result since no new input data is available from the previous stage.

The display stage of the pipeline displays frame *n* with the results from the most up-to-date rendering. All the stages happen to line up fairly well for frame *n*, and image *i* delay is not much more than the sum of the individual stage components. Frame *n*+1 repeats the display of an entire frame because no new data is available when starting to display that frame. Frame *n*+1 has a delay of an additional frame time more than the image *i* delay. Frame *n*+4 is delayed even further due to similar reasons. No redundant data is computed for frame *n*+5, but image *i*+2 delay is quite large because the rendering and application stages must complete their previous computations before starting new computations.

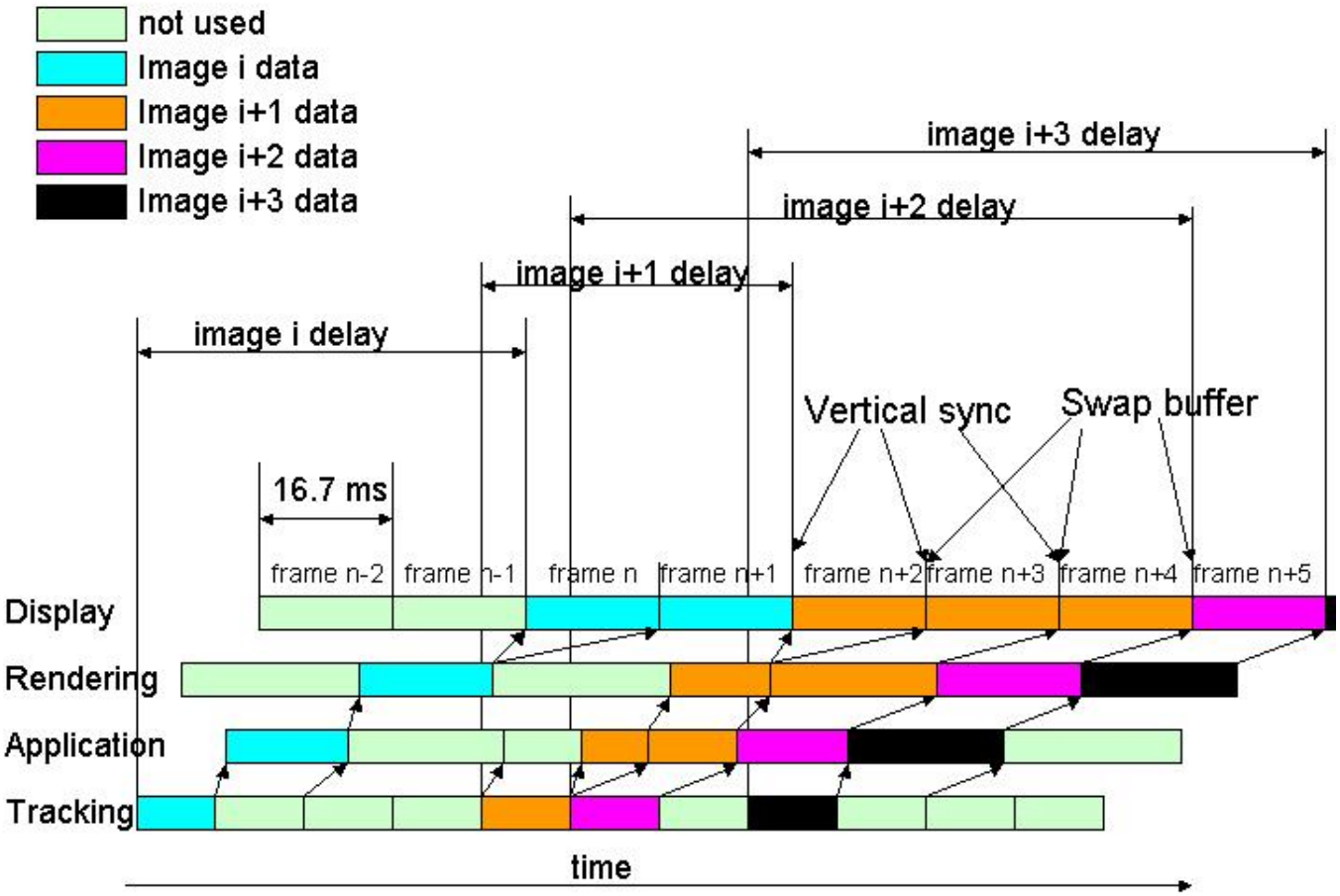

Figure 6. A timing diagram for a typical VR system. In this non-optimal example, the pipeline stages execute concurrently but stages must wait for previous stages to provide new information in order to compute new information themselves.

4.4 Reducing Synchronization Delays

It is evident from the above timing analysis that reducing system delay and delay variability is not a simple matter of optimizing individual stages, since synchronization delay can be large and vary considerably. Fortunately, simple changes to standard VR systems can greatly reduce synchronization delay.

[Bryson and Johan 1996] dramatically reduce system delay for systems with a long application stage. The application does not necessarily need to know about tracking. The rendering stage can rely on transformations directly from the tracking stage, thus bypassing the application stage. The renderer uses the most up-to-date application data, which is independent from tracking, even though that data can be quite old. Therefore, depending on the coupling between components, this application stage may add little or even no delay. This

allows immediate natural viewer interaction through head movement even though the actual application may be updating at a much slower rate. Bryson and Johan describe an example of a simulated wind tunnel where the simulation (i.e., the application) is independent and updates at a slower rate than the rest of the system. For complex simulations, the wind vectors may take several seconds to compute the next step of the simulation. However, the user is free to examine the static steps of the simulation from different points of view at interactive rates.

Pipeline stages can often be divided into independent sub-steps to reduce delay. This can greatly decrease synchronization times by reducing wait times. For example, waiting to swap on vertical sync (as described in Section 3.5, Display) adds a large amount of delay and delay variance since the entire frame must be scanned out before swapping buffers. If the frame can be broken into smaller steps by not waiting to swap on vertical sync, then delay and its variability are greatly reduced. The entire frame takes the same amount of time to display (the frame time), but each individual component of that frame uses the most up-to-date rendered image.

Figure 7 shows an example of reducing system delays. The system does not wait to swap on vertical sync. The application is executed asynchronously from the rest of the system. Even though the rendering times are relatively large, a typical latency is not much larger than the contributing component sums because wait times are short. As the rendering times get smaller, the system delay decreases and a just-in-time pixels solution is approached as described in Section 3.5, Display.

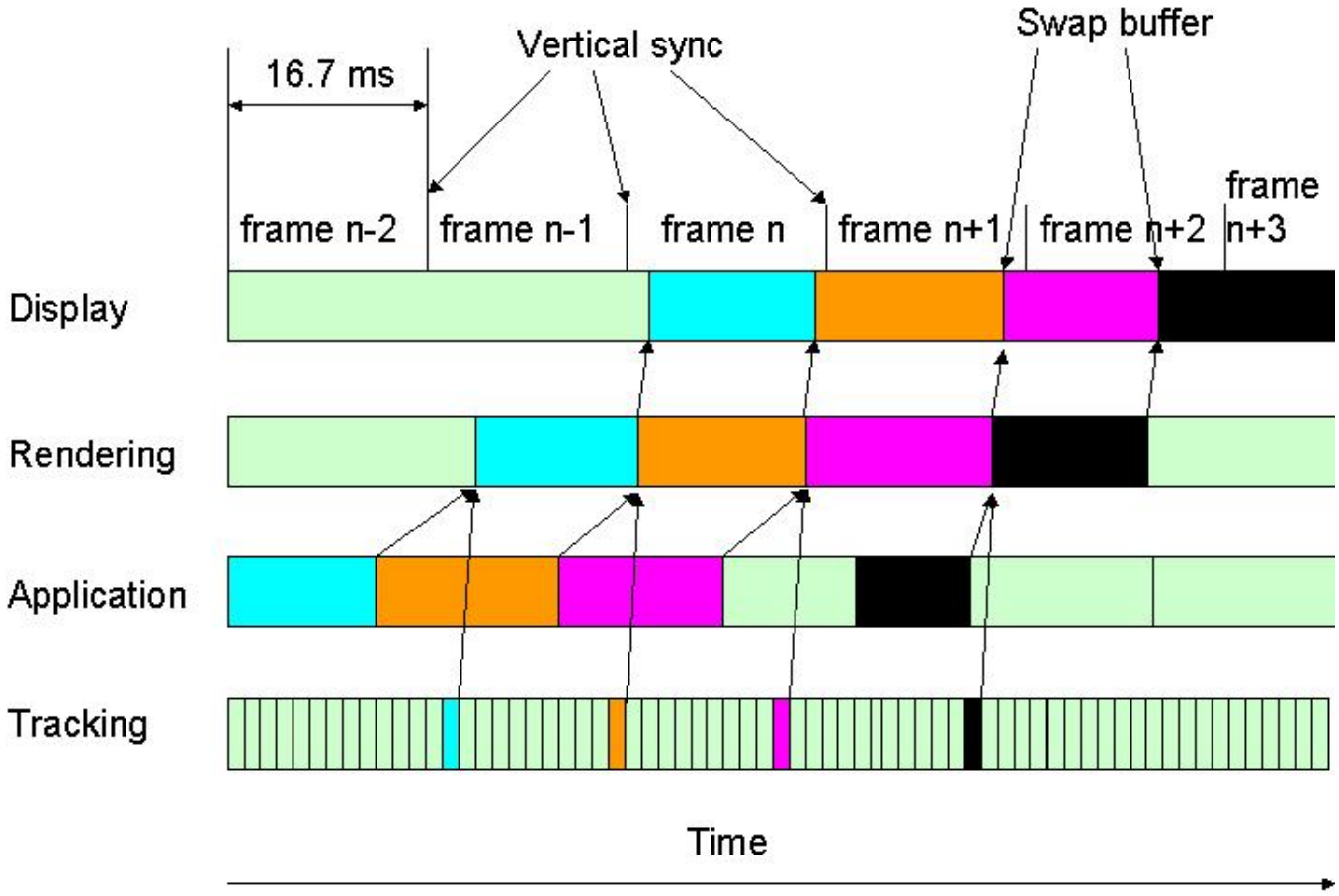

Figure 7. Timing diagram with small synchronization delays. The application is executed asynchronously and does not wait to swap on vertical sync.

5 PREDICTION

Perfect prediction would allow a system to have zero effective latency. Unfortunately, predicting human head motion is impossible due to human non-determinism. However, for small prediction intervals and small angular head frequency, a system can predict reasonably well where the user will look. The prediction interval p is equal to the estimated system delay. ω is the angular frequency and is proportional to how often the user changes direction per second. The acceptability of p depends on the value of ω, implementation,

operator task, user tolerance of error, etc. If p initially does not produce satisfactory results, then one can iteratively reduce p until adequate as discussed in the previous section (Section 4, System Analysis and Optimization).

5.1 Prediction Error

Prediction error increases rapidly with increasing p and increasing ω even with perfect noise-free tracking inputs [Azuma and Bishop 1995]. For prediction with a second order polynomial (i.e., position, velocity, and acceleration) the rate of error growth is roughly $p^2\omega^2$. One result of this error appears as visual jitter, which is the magnification of high frequency signals or noise. Users perceive this visual jitter as shaking of the display.

If p is not equal to system delay, then the system will predict for the wrong time. Since consistent delay can be difficult to obtain in practice, it may be possible to continually monitor system delay and set p to the recent average system delay. Unfortunately, precise timing measurements can be difficult to obtain in real-time with today's systems.

Most trackers provide six degree of freedom (DOF) data (position and orientation). Numerical differentiation allows estimation of velocity and acceleration from this six DOF data. Whereas velocity and acceleration data generally improve prediction, derivative estimation from discrete data is not precise and accentuates noise. Computing acceleration through double differentiation results in even greater noise. Thus, reading velocity and acceleration directly from inertial based devices is a more accurate method.

5.2 Inertial-based prediction

[Azuma and Bishop 1994] found in their study that inertial-based prediction (i.e., prediction based on gyro and accelerometer readings) is 2-3 times more accurate than non-inertial-based prediction and 5-10 times more accurate than no prediction (for p equal to 60 ms).

Rate gyroscopes directly measure angular velocity. This greatly reduces the error in extrapolation because there is no need to differentiate discrete data. The angular velocities from the gyroscopes can be integrated over time to obtain change in orientation. This integrated orientation has a low-frequency drift caused by integration error buildup, but when integrating over short times drift is insignificant.

Accelerometers can directly measure linear acceleration. However, the improvement in prediction is small compared to using gyroscopes, since translation error is typically minor compared to orientation error.

6 POST-RENDERING IMAGE CORRECTION

This section starts by describing environment mapping, which improves performance by mapping images onto a cube surrounding the user, and concludes with a simplified version that corrects yaw error with fast scanline shifts.

6.1 Environment mapping

Environment mapping allows quick viewing of complex worlds by projecting the world onto the six sides of a large cube surrounding the user [Greene 1986]. Unfortunately no motion parallax is possible with environment maps. However, for objects at a reasonable distance from the user, small translations result in little registration error. Environment mapping can be optimally implemented by recognizing that rotations cause every pixel to shift location. Head rotation simply alters what part of display memory is accessed — no other computation is required [Regan and Pose 1994].

6.2 Image Transformations

Environment mapping can be further simplified by projecting the world onto a single image plane instead of a cube. This technique works well if the user is looking in the same general direction as the original projected image plane. After projection, small movements can occur after projection with no perceived error.

Much of the latency due to rendering can be reduced by using a two-pass algorithm. The first pass, which may take several milliseconds, renders the scene to a larger-than-screen-sized polygon. The second pass then renders the polygon from the current viewpoint to account for the current viewpoint. Since only a single polygon is rendered, this second pass is quick and occurs in constant time independent of scene complexity.

The second pass can be further sped up by shifting the image instead of re-rendering from the current viewpoint. Small user yaw and pitch rotations can be approximated after projection with image shifts. Image shifting is unable to correct for roll, but fortunately error due to roll is minor compared to error due to pitch and yaw. The projected image must be larger than the final displayed image in order for subset selections to yield enough pixels to fill the entire display device.

The Reflex HMD [Kijima *et al.* 2001] implements a horizontal and vertical image correction in hardware. This system renders an image with a larger field of view than the HMD. A fast gyroscope determines what part of the larger image should be extracted just before display, effectively shifting the entire image. Their system uses a LCD for display.

One can go beyond image shifting by shifting individual scanlines through frame buffer re-addressing. This can almost completely remove yaw error due to display delay. The display processor can render a larger image than the display device can display. Scanlines can then be individually extracted, instead of the entire frame, based on the current orientation.

7 PROPOSED METHOD

I have described problems due to latency and several methods of reducing latency. To overcome the limitations of these methods, I now propose a hybrid method combining system analysis and optimization, prediction, and post-rendering image correction that yields a system with approximately zero latency. The method concentrates on reducing orientation error, which is the largest source of registration error. Translation error is small for large environments where objects are at a distance [Regan and Pose 1994]. Even for closer objects, error due to motion parallax is small compared to rotational errors for typical user movements.

7.1 Method Overview

First one reduces system delay and delay variance to a reasonable level using the SPE methodology. A prediction interval *p* below 80 ms (i.e., system delay) is a good initial goal [Azuma and Bishop 1994]. Using gyroscopes in addition to standard tracking, the system predicts three DOF orientation for the time when the image will be displayed and renders a large image for that view. Post-rendering image correction then corrects the yaw error due to imprecise prediction. The system performs this correction every scanline by updating the current yaw orientation from the gyroscope, calculating the error, and selecting the appropriate starting scanline address from the large rendered image. The gyroscope read, error calculation, and scanline shifts are implemented in specialized hardware and occur less than 0.1 ms before scanout, allowing the latest head orientation to be used for selecting appropriate pixels from the larger image.

If results are unacceptable, the SPE methodology is repeated (reducing system delay) until latency-related errors and their effects are acceptable.

7.2 Component relationship

All steps of the proposed method are related, and the system must be integrated carefully because the timing and accuracy of preceding steps non-linearly and dramatically affect the size of error in succeeding steps.

Small prediction intervals require only small system delay. Since error goes as the square of the prediction interval, smaller prediction intervals result in much less error. These smaller errors can be corrected for with post-rendering image correction (for yaw orientation). If early steps contain too much error, then that error becomes difficult to correct in later steps. This relationship can also be thought of in reverse — the better the post-rendering image orientation correction then the less accuracy is required of the predictor, which allows for more system delay. However this relationship is only a square-root relationship.

Figure 8 shows this reduction in error. If the final resulting error is not small enough, then the SPE method is repeated until requirements are met.

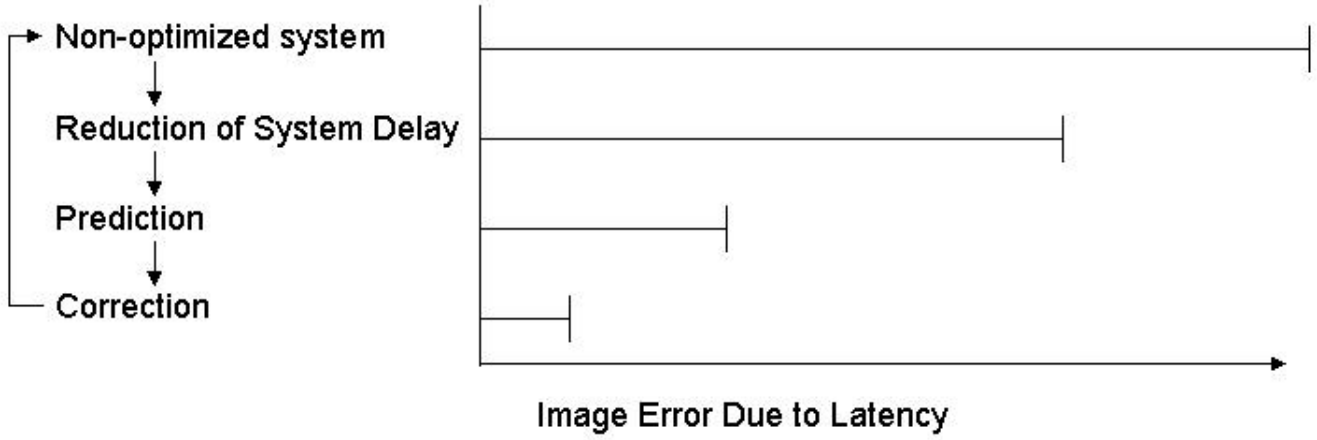

Figure 8. Error due to latency is decreased in each step of the process. If the error after correction is not acceptable, the SPE method is repeated until requirements are met.

7.3 Implementation Details

Figure 9 shows a block diagram of the proposed solution. The PC, Tracker and CRT are standard components of typical VR systems. Additional components required for the proposed system are three gyroscopes (one for each axis) and a specialized image-correction chip (ICC).

Angular velocities are determined from the gyroscopes and are read by both the predictor (contained in the PC) and the ICC. The predictor uses angular velocities from all three gyroscopes, but the ICC uses only yaw velocity. The predicted angle offsets are simply the current angular velocities multiplied by the system delay *p*. The predicted yaw angle offset α is sent to the ICC. Note the figure shows the α line conceptually, but in reality α is sent as a pixel in the DVI signal. In addition, a time signal is sent via the parallel port from the PC to

the ICC. This time signal informs the ICC when to start integrating angular velocities to determine the integrated angular offset β, where β is updated every scanline. The difference between α and β is the error Δ and is proportional to the number of pixels that need to be shifted.

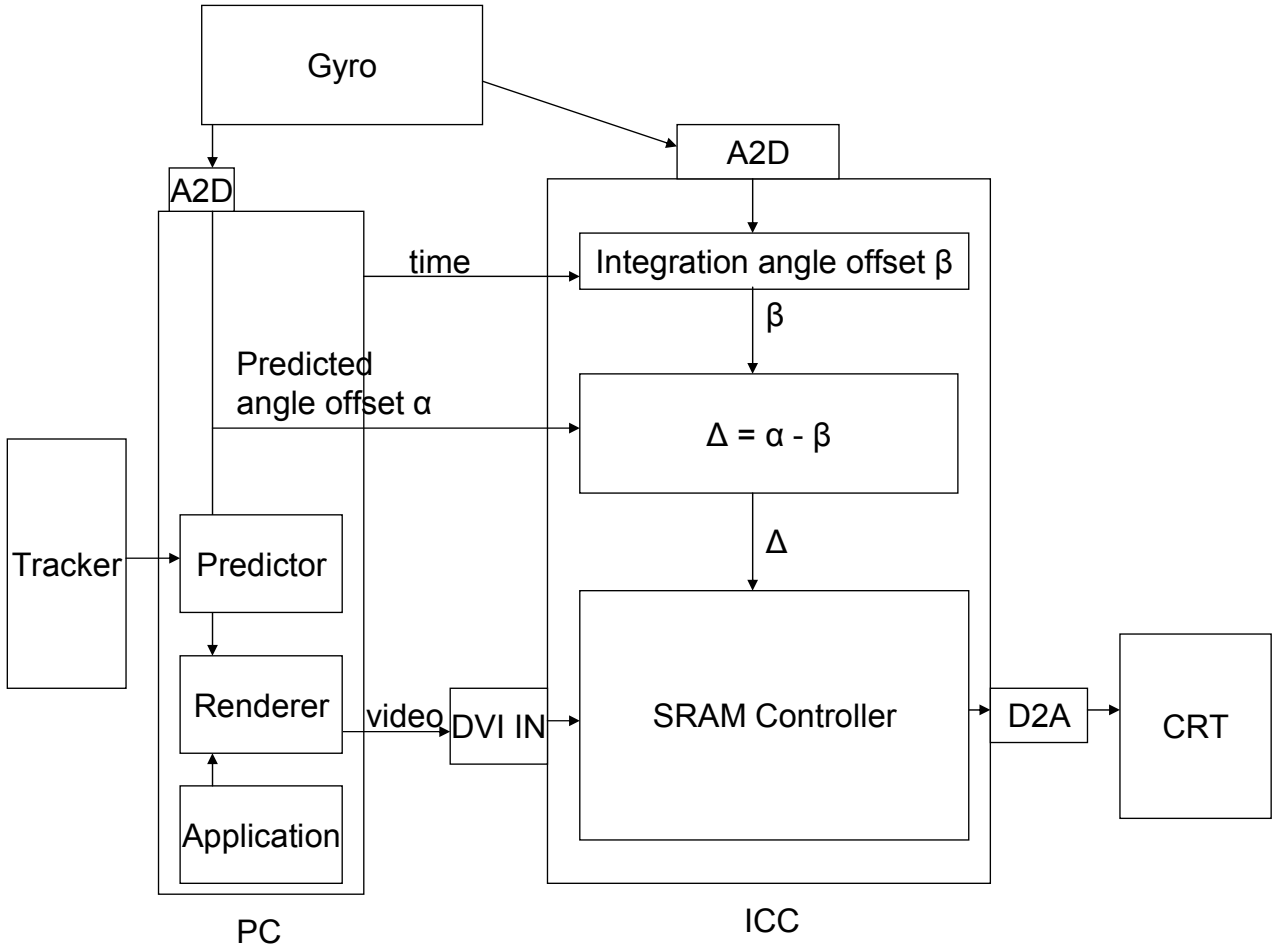

Figure 9. Block diagram of the proposed system.

7.4 Results

The end result of the proposed method as perceived by the user is near-zero latency. Yaw orientation error, which is the largest type of error, is especially reduced by the post-rendering image correction.

Figure 10 shows a timing diagram of the proposed system. The application stage is not shown, since the rendering stage receives transformations directly from the tracker. The majority of the system delay in this example comes from rendering. Prediction and correction easily compensates for this small system delay.

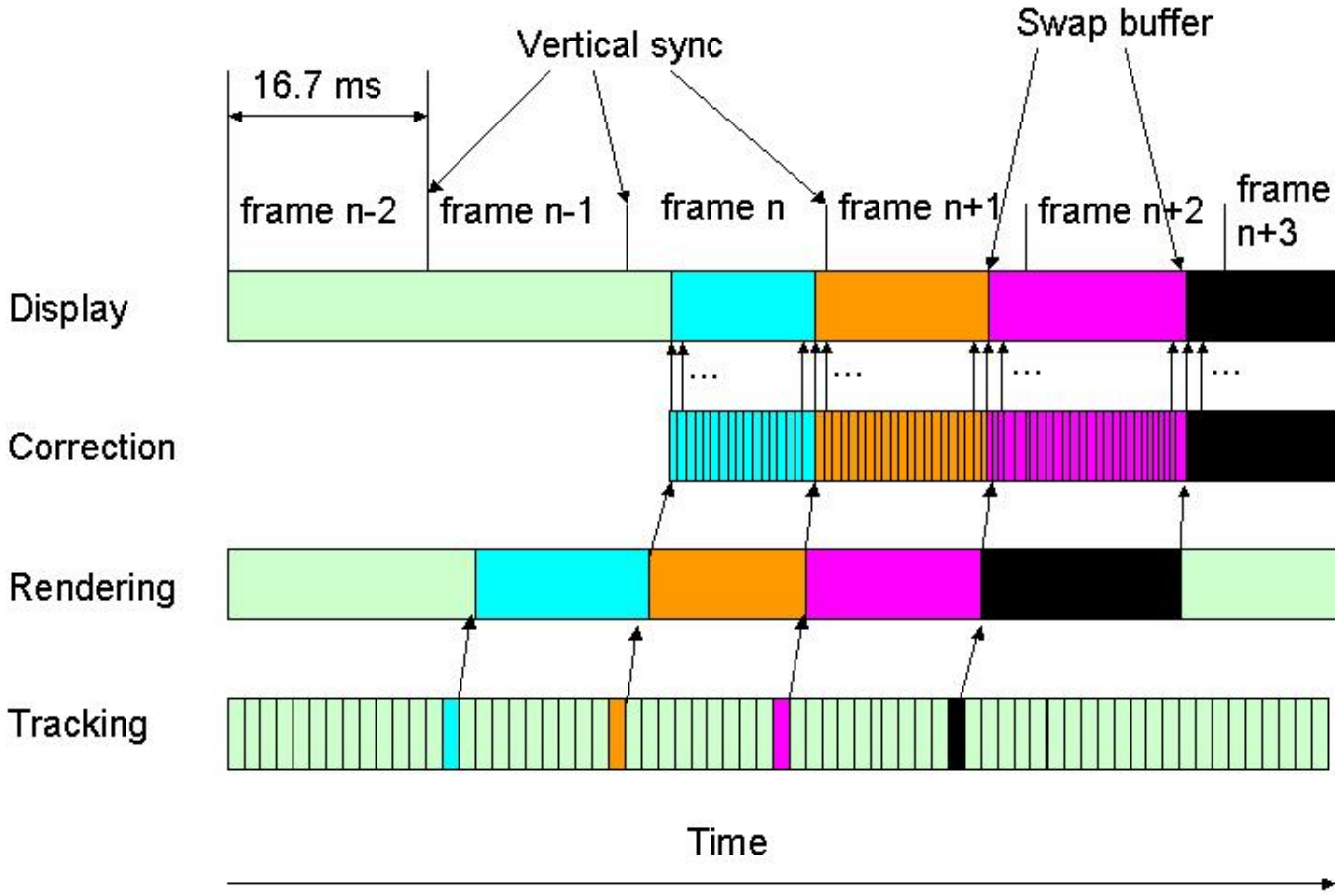

Figure 10. Timing diagram of the proposed system.

Figure 11 shows three versions of a frame (showing the same static virtual block as in Figure 3) composited together as the user is turning the head right-to-left. The figure compares a frame that waits on vertical sync, a frame that does not wait on vertical sync, and the proposed hybrid solution with scanline correction. As in Figure 3, the four larger regions in Figure 11 represent renderings for the predicted point of view at different points in time.

The skewed lines represent shifted scanlines of the rectangular object. The rectangular object appears similar to a parallelogram on paper but appears as a rectangle to a user looking right-to-left in a dynamic display due to the scanlines being displayed at different points in time. Instead of having four large tears, there are many small tears; the tears are not as evident since each tear is small. This correction shifts individual scanlines to correct for differences in time. The result is just-in-time scanlines, which reduces display latency without adding disturbing tearing artifacts.

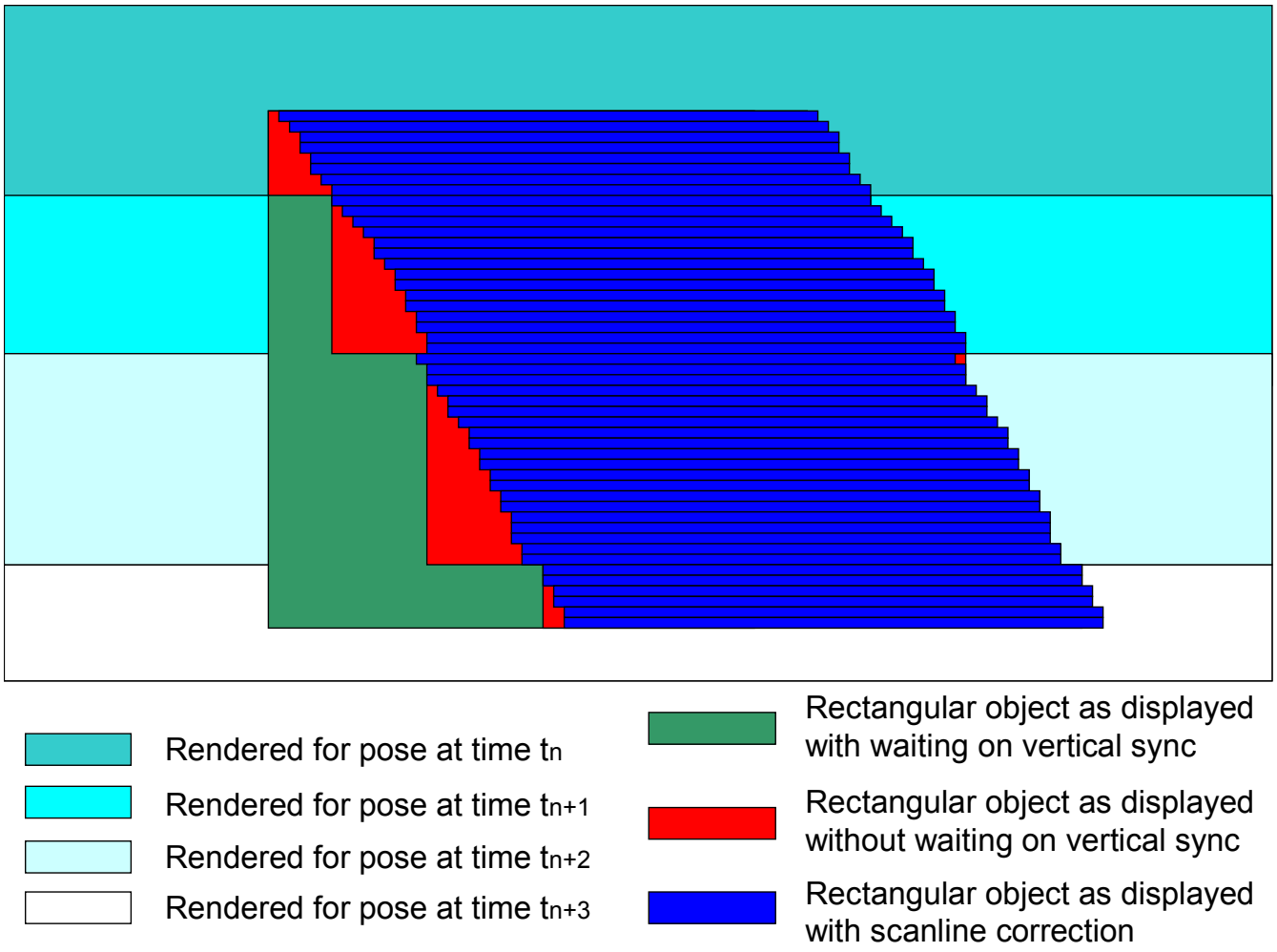

Figure 11. Output image with shifted scan lines.

8 CONCLUSIONS AND FUTURE WORK

Several subtopics of computer science have been reviewed in order to understand and reduce ill effects caused by latency. Studying these issues leads to a hybrid system, which should result in reduction of the most critical types of error due to system delay. The proposed solution combines system analysis and optimization, prediction, and post-rendering image correction.

The human limits of latency perception are not yet known, since no such system has reduced latency to such a level. To determine these human limits, a system must first be built that is capable of reducing latency below the noticeable threshold and then user studies must be conducted to determine requirements of VR. Since the proposed method concentrates on the most common type of dynamic errors (i.e., horizontal errors caused by yaw rotations), other dynamic errors may become more apparent as the horizontal error is reduced.

Vertical error could be reduced, in a way similar to horizontal error correction, by shifting scanlines vertically. Unfortunately, vertical scanline shifts require an additional frame time of delay and more sophisticated post-rendering hardware. Tracker position data and/or accelerometers can help with prediction of position but cannot help with post-rendering 2D image correction. If full translation correction is needed, then a full 3D warp is required [Mark *et al.* 1997]. 3D transformations of depth images cause visual artifacts that appear as 'holes' due to motion parallax. In many situations, these artifacts can be greatly reduced if prediction is done well.

The proposed system is currently in the process of development, and it remains to be seen how such a hybrid system compares with existing systems. The error due to other unsolved problems (calibration, distortion, etc.) may outweigh dynamic errors for smaller system delays. Regardless, latency and dynamic registration will continue to be problematic in the foreseeable future.

9 ACKNOWLEDGMENTS

The UNC Effective Virtual Environments research group has been crucial to this work. Insight and reviews from Anselmo Lastra, Fred Brooks, Mary Whitton, Herman Towles, Henry Fuchs, Neeta Nahta, Sharif Razzaque, and Susan Jerald are greatly appreciated. This research has been supported in part by a National Physical Science Consortium Fellowship and by stipend support from HRL Laboratories.

10 COMPUTER SCIENCE SUBTOPIC REFERENCES

Hardware Architecture. WHITTON, M. C. 1984. Memory Design for Raster Graphics Displays. *IEEE Computer Graphics and Applications,* 4(3), pp. 48-65.

Software Engineering. SMITH, C. U. 1986. The Evolution of Software Performance Engineering: A Survey. in Proceedings of *1986 Fall Joint Computer Conference*, pp. 778-783.

Human Computer Interaction. ALLISON, R. S., HARRIS, L. R., JENKIN, M., JASIOBEDZKA, U., and ZACHER, J. E. 2001. Tolerance of Temporal Delay in Virtual Environments. in Proceedings of *IEEE Virtual Reality 2001*, pp. 247-254.

11 REFERENCES

3rdTech (2002) "Hiball-3000 Wide-Area Tracker," User Manual.

Allison, R. S., Harris, L. R., Jenkin, M., Jasiobedzka, U., and Zacher, J. E. 2001. Tolerance of Temporal Delay in Virtual Environments. in Proceedings of *IEEE Virtual Reality 2001*, pp. 247-254.

Azuma, R. T. and Bishop, G. 1994. Improving Static and Dynamic Registration in an Optical See-through Hmd. in Proceedings of *SIGGRAPH'95*, pp. 197-204.

Azuma, R. T. and Bishop, G. 1995. A Frequency-Domain Analysis of Head-Motion Prediction. in Proceedings of *SIGGRAPH '95*, pp. 401-408.

Bryson, S. and Johan, S. 1996. Time Management, Simultaneity and Time-Critical Computation in Interactive Unsteady Visualization Environments. in Proceedings of *IEEE Visualization '96*, pp. 255-262.

Ellis, S. R., Young, M. J., Adelstein, B. D., and Ehrlich, S. M. 1999. Discrimination of Changes in Latency During Head Movement. in Proceedings of *Computer Human Interaction*, pp. 1129-1133.

Greene, N. 1986. Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics and Applications,* 6(11), pp. 21-29.

Holloway, R. L. 1995 *Registration Errors in Augmented Reality Systems*, Ph.D. Dissertation, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA.

Kijima, R., Yamada, E., and Ojika, T. 2001. A Development of Reflex Hmd - Hmd with Time Delay Compensation Capability. in Proceedings of *2nd International Symposium on Mixed Reality (ISMR2001)*, pp. 172-179.

Mark, W. R., McMillan, L., and Bishop, G. (1997) "Post-Rendering 3d Warping." In *Proceedings of the 1997 Symposium on Interactive 3d Graphics*, Providence, RI, pp. 7-16.

Meehan, M., Brooks, F., Razzaque, S., and Whitton, M. 2003. Effects of Latency on Presence in Stressful Virtual Environments. in Proceedings of *IEEE Virtual Reality 2003*.

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*.

Miné, M. and Bishop, G. 1993 "Just-in-Time Pixels, " Technical Report, Report No. TR93-005, University of North Carolina at Chapel Hill.

Miné, M. R. 1993 "Characterization of End-to-End Delays in Head-Mounted Display Systems, " Technical report, Report No. TR93-001, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC.

Nakanishi, K., Takahashi, S., Oura, H., Matsumura, T., Miyake, S., Kobayashi, K., Oda, K., Tahata, S., Yuuki, A., Someya, J., Yamakawa, M., and Gofuku, E. 2001. A Fast-Response 15-In. Xga Tft-Lcd with Feed-Forward Driving (Ffd) Technology for Multimedia Application. in Proceedings of *Society for Information Display 2001*, pp. 488-491.

Olano, M., Cohen, J., Miné, M., and Bishop, G. 1995. Combatting Rendering Latency. in Proceedings of *Symposium on Interactive 3D Graphics*.

Regan, M. and Pose, R. 1994. Priority Rendering with a Virtual Reality Address Recalculation Pipeline. in Proceedings of *SIGGRAPH 94*, pp. 155-162.

Smith, C. U. 1986. The Evolution of Software Performance Engineering: A Survey. in Proceedings of *1986 Fall Joint Computer Conference*, pp. 778-783.

So, R. H. Y. and Griffin, M. J. 1995. Effects of Lags on Human Operator Transfer Functions with Head-Coupled Systems. *Aviation, Space and Environmental Medicine,* 66(6), pp. 550-556.

Sutherland, I. E. 1968. A Head-Mounted Three Dimensional Display. in Proceedings of *Proceedings of the 1968 Fall Joint Computer Conference, AFIPS Conference Proceedings*, pp. 757-764.

Whitton, M. C. 1984. Memory Design for Raster Graphics Displays. *IEEE Computer Graphics and Applications,* 4(3), pp. 48-65.