An Event Architecture for Distributed Interactive Multisensory Rendering

Timothy Edmunds*
Rutgers University

Dinesh K. Pai†
Rutgers University

ABSTRACT

We describe an architecture for coping with latency and asynchrony of multisensory events in interactive virtual environments. We propose to decompose multisensory interactions into a series of discrete, perceptually significant events, and structure the application architecture within this event-based context. We analyze the sources of latency, and develop a framework for event prediction and scheduling. Our framework decouples synchronization from latency, and uses prediction to reduce latency when possible. We evaluate the performance of the architecture using vision-based motion sensing and multisensory rendering using haptics, sounds, and graphics. The architecture makes it easy to achieve good performance using commodity off-the-shelf hardware.

Keywords: multimodal interaction, augmented reality, multisensory, synchronization, latency, mixed reality, virtual environments

1 INTRODUCTION

In augmented reality (AR) applications, the user's productivity and sense of seamlessness can be destroyed by *latency* between the user's input and the system's response, and *asynchrony* between the system's different output sensory modalities. This is particularly important in multisensory environments in which it is necessary to control and synchronize a large number of specialized sensors and display devices, each controlled by different computers over a local area network.

A well known approach to reducing latency and asynchrony is to make predictions about the future interactions between the user and the system. The limitations of the human body make it possible to perform two distinct types of prediction: *continuous* and *discrete*. Prediction of continuous future state is possible because inertia and muscle dynamics cause human movement to be relatively smooth; once a body part is in motion, it tends to continue that motion. For example, a virtual golf simulation might predict the trajectory of a golf swing in order to graphically render the motion of the club.

The second type of prediction that can be used to improve latency and synchronization is the prediction of discrete interaction events. Many VE and AR applications involve discrete decisions by the user; after the decision is in the user's mind, the body takes time to physically manifest the decision. Following the virtual golf example above, the application might not be concerned with the precise trajectory of the golf club throughout the user's swing; instead, the important aspect of the interaction is the moment when the club makes contact with the ball, triggering the rendering (e.g., graphically, acoustically, and haptically) of the ball's launch. Correctly predicting this event, rather than the entire club trajectory, is likely to be both more important (given that the motion of the club is a perceptual blur), and simpler (since the speed of a golf swing would require a very high frequency tracking sensor to yield enough readings to accurately predict the entire trajectory).

*e-mail: tedmunds@cs.rutgers.edu
†e-mail:dpai@cs.rutgers.edu

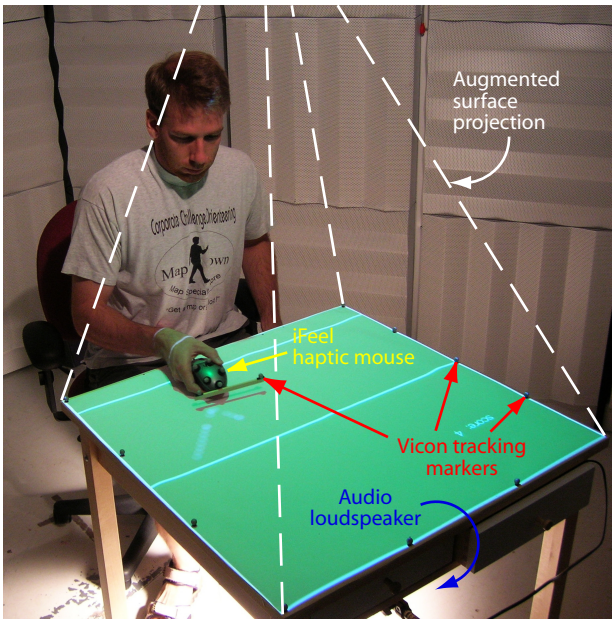


Figure 1: Multisensory PONG! This simple game demonstrates the issues arising in many distributed interactive virtual environments. The user input is measured (using an optical motion capture markers), and the game logic has to render the virtual world in time, with graphics from a projection display, sound from loudspeakers, and haptic forces from a force feedback mouse.

The ability to generate advanced warning of user actions prompts us to develop an event based architecture for interactive applications.

When considering discrete events in AR, other aspects of the human user affect the design decisions. Specifically, characteristics of the user's perception affect the interpretation of output stimuli as cohesive events. Psychophysical studies yield intermodal and intramodal thresholds beyond which stimuli are perceived as separate events (see § 2). These thresholds impose design requirements for the synchronization of output rendering across sensory modalities.

Our Contribution is a flexible, easy-to-implement distributed architecture for rendering multisensory spatio-temporal events with low latency and good synchronization. Our approach decouples the prediction of system state from synchronization of multisensory output, and can be run on commodity consumer level computers and networks. We show that structuring multisensory AR applications within an event-based architecture simplifies the problem of reducing both the relative latency between the rendering of multisensory aspects of the same event, and the end-to-end latency between the occurrence of the event and its perception by the user.

The rest of this paper is organized as follows: in § 2 we discuss related work, including some psychophysical results that influence our design choices. § 3 contains an analysis of latency and asynchrony in an interactive application pipeline. We present our general-purpose architecture in § 4. In § 5 we give a proof-of-concept evaluation of how our architecture can be used to achieve

latency bounds imposed by human perception. Finally, in § 6, we describe a concrete example implementation that illustrates how our architecture can be used to develop multi-modal spatio-temporal event based interactive applications.

2 Related Work

Many papers proposing architectures for distributed interactive applications present specific codebases, APIs, or communications protocols [32, 4, 16, 30, 29, 25, 8, 21]. Our intent is to provide a design pattern for distributed multisensory interactive applications that can be flexibly applied in a manner independent of the underlying application development environment, and that can be easily incorporated in mixed model architectures that could allow a broader class of applications [17].

The purpose of our architecture is to address two key issues: system latency (the delay between a user signalling a decision and the system rendering the appropriate response), and cross-modal synchronization (the temporal separation between different aspects of a single event being rendered). The impact of these two issues on interactive applications is well-recognized [9, 18, 17, 21, 4], and a number of methods to address them have been proposed in the context of VE and AR applications.

Most of the work on distributed interactive applications (like shared virtual worlds [20, 16, 21, 30, 25] and cooperative workspaces [5, 29]) focusses on accurately replicating state between different sites, where a different user (or set of users) experiences each "subjective" [26] interpretation of the shared state. In multimodal applications, we must be concerned with the replicating a single subjective interpretation of the state across different sensory modalities.

The coordination of different sensory modalities at a single site is usually left to be handled by a single powerful machine [25], likely making use of multiprocessor hardware [4]. However, building multimodal systems often requires (or is simplified by) separate machines to control independent hardware for some or all of the sensors or rendering systems, and a desirable architectural feature is the ability to connect different local components of the application over a network.

Distributing replicated state over a network is the goal of many of the approaches mentioned above, but the techniques used to address replication between different users are often unsuitable for replication between different aspects of a single user's experience. For instance, tradeoff between local responsiveness and the frequency of replication inconsistencies [21] addresses the need for replication correctness, but comes at the cost of increased end-to-end latency; when dealing with AR, this cost is particularly problematic, since the real world cannot be time-shifted, and always provides a baseline against which other latencies are measured. Relaxing the synchronization by allowing temporary or slight inconsistencies [8] can result in conflicting cues in the user's perception; the applicability of such constraint relaxation approaches is bounded by human perceptual characteristics.

Previous work in the AR field has recognized the impact of latency on the user's experience [22], and the various contributions to latency found in typical AR applications have been categorized [22, 35, 12]. These works have also established the importance of latency measurement as the precursor to latency elimination. However, the analysis and solutions presented in those papers was set in the context of single machine, single output modality AR applications, whereas we are also concerned with multisensory output applications (particularly the case where different machines are producing different components of the output). The incorporation of multiple output modalities adds new sources of lag to the latency analysis (and new psychophysical constraints to the latency requirements). The desire to host AR applications on locally distributed systems also calls for further techniques for latency reduction.

When rendering multiple simultaneous signals that are intended to be perceived as a single stimulus, we must quantify what temporal range qualifies as "simultaneous". Psychophysical studies have shown that the modality of the signals affects the temporal threshold within which simultaneity is perceived. Whereas two sounds must occur within 2 ms of each other to be perceived as a single sound [14], the sight/sound asynchrony threshold has been measured at values ranging from as high as 175 ms [23] to as low as around 75 ms [14, 7]. The touch/sound threshold is approximately 24 ms [1, 6], and the touch/sight threshold is approximately 45 ms [34] (but has been seen to vary with attention). These perceptual thresholds dictate the synchronization that must be maintained between output renderers to provide a consistent experience by the user.

Even as LAN speeds and clock synchronization algorithms [27] advance to approach the behaviour of a multiprocessor machine, the sensors and rendering devices used impose a minimum end-to-end latency from direct signal detection to system response. Advances in sensor and rendering hardware can lower this minimum [28], but with current and foreseeable systems, there is an unavoidable latency floor that establishes a need for the prediction of future state.

A major application for the prediction of future state is head tracking for use in immersive VR [9] and head-mounted AR [2]. When determining user head orientation for AR, latency results in registration errors during movement [2, 3]; in VE, the visual artifacts due to latency can cause performance degradation, loss of immersion, and nausea [31, 18]. Predictive compensation (constantly estimating real current head orientation from old measurements) is a widely explored [10, 13, 15, 19, 3] remedy to this problem.

This type of continuous state prediction is the one most commonly seen in the literature. In contrast, our approach is to focus on the prediction of when and where discrete system events (including user input) will occur (this approach was introduced in [24]). The focus on discrete events rather than on continuous state highlights different issues. For example, continuous prediction is robust with respect to the occasional spurious prediction (which will only elicit a momentary blip in the user's perception) whereas with discrete prediction, it is more important that every prediction be (close to) correct. Slight errors in the time/location at which an event is predicted will likely to go unnoticed by the user (whereas even small persistent errors in continuous head tracking will cause perceptible registration problems).

3 Analysis of Latency and Asynchrony

The goal of our event prediction architecture is to eliminate the latency between real-world events and the synthesized response to them, and the asynchrony between different modalities in responding to the same event. To understand our approach to achieving this goal, we must examine the factors that contribute to the existing latency.

Our model of the data-flow within the system is that state data flows from sensors to controllers, where events are detected/predicted and dispatched to renderers. This model can be illustrated by charting the time-line of one piece of data as it flows through a naïve implementation of the system (see Figure 2(b)).

Let t_{event} be the time when a renderable event occurs, at which point the world is in state s_{event}. Let $t_{in} = t_{event} + \delta_{in}$ be the time at which the sensor begins processing the s_{event} data (δ_{in} includes both the time it takes for real world phenomena to reach the sensor and the processing overhead involved in such activities as sampling, buffer reading, etc.). The sensor then processes the s_{event} data (for example, an optical tracking sensor might perform 3-D reconstruction, temporal filtering, etc.), and produces a sensor reading, r_{event} at $t_{sensed} = t_{in} + \delta_{sense}$; for the purposes of

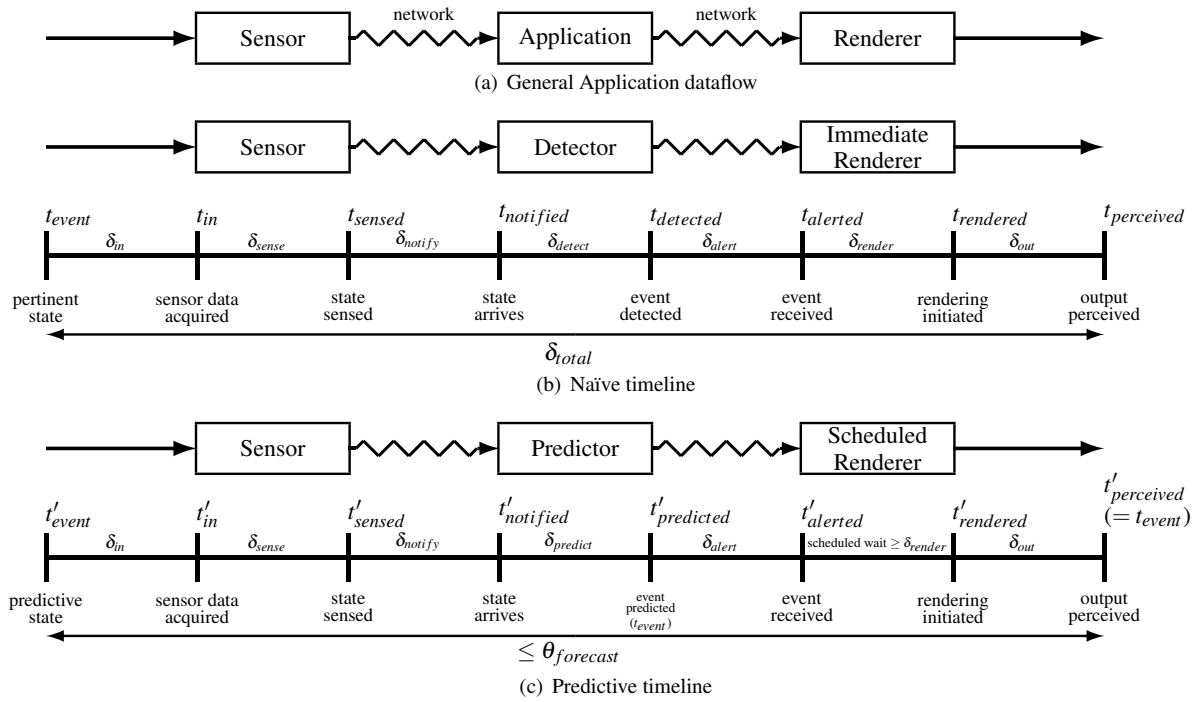


Figure 2: (a) In general, an interactive application has a dataflow from the real world, through a sensor, through the application logic, then through a renderer, and finally back through the real world to the user. (b) Each component of the latency in the naïve event-based application contributes directly to the end-to-end latency. (c) In the predictive event-based application, the timeline begins at an earlier state from which the event can be predicted; predicting the correct time to initiate rendering results in an output perception time that coincides with the real world pertinent state.

this discussion, the t_{sensed} is the value timestamped onto the sensor reading. r_{event} (and timestamp) is sent (e.g., over the network) to the controller, arriving at time $t_{notified} = t_{sensed} + \delta_{notify}$. In the naïve system, the controller checks to see whether r_{event} implies that a renderable event has occurred; if it has, an event is dispatched (at time $t_{detected} = t_{notified} + \delta_{detect}$) to the renderer. This overhead δ_{detect} could include the computation necessary to estimate hand configuration from pressure sensor data in order to do gesture detection. The event arrives (over the network) at time $t_{alerted} = t_{detected} + \delta_{alert}$, and (in the naïve system) the renderer "immediately" (at time $t_{rendered} = t_{alerted} + \delta_{render}$) initiates rendering of an appropriate stimulus, which is perceived by the user at time $t_{perceived} = t_{rendered} + \delta_{out}$. In addition to real-world transmission from the output device to the user's sensory organs, δ_{out} includes overhead added by the operating system, the device drivers, and the hardware itself. The end-to-end latency for the naïve system is thus $\delta_{total} = \delta_{sense} + \delta_{notify} + \delta_{detect} + \delta_{alert} + \delta_{render} + \delta_{out}$. This means that the rendered stimulus will be perceived δ_{total} too late.

As developers of an application using a given set of sensors and rendering devices, we only have control over certain parts of this time-line. Even if the controller and render applications are implemented with no computational or communication overhead (i.e., $\delta_{detect} = \delta_{alert} = \delta_{render} = 0$), there will still be some latency ($\delta_{in} + \delta_{sense} + \delta_{notify} + \delta_{out}$) between an event occurring and the user perceiving the system's response to the event. This shows that rendering of an event must be initiated *before* the controller is notified of the sensor reading corresponding to the event; i.e., the event must be predicted from previous states reported by the sensor.

We classify the identified sources of latency into two categories: those which can be *circumvented*, and those which must be *compensated for*. Recall that we defined t_{sensed} to be the time with which the sensor reading is stamped. If all the application's machines are working to a common clock, δ_{notify}, δ_{detect}, δ_{alert}, and δ_{render} can all be circumvented by prediction; if (from r_{event}) we can predict the event at least $\delta_{circumvent} = \delta_{notify} + \delta_{detect} + \delta_{alert} + \delta_{render}$ in advance, they will not affect the perceived end-to-end latency. However, since δ_{in}, δ_{sense}, and δ_{out} represent latencies before time-stamping and after rendering, they will be manifest no matter how far in advance the event is predicted. These latencies must be compensated for by offsetting the rendering time.

4 Spatio-Temporal Event Architecture

4.1 Programming Model

Our model of a multisensory interactive system incorporates three entities: sensors, controllers, and renderers (Figure 3). The sensors continually send *time-stamped* measurements to the controllers. The controllers decide what spatio-temporal events are renderable and *predict* when and where these events will occur. The event predictions are sent to renderers that *schedule* and manifest appropriate output for the event. The important point is that this decouples the latency (which depends on how prediction is accomplished) from synchronization (which depends on the accuracy with which the clocks on different machines are synchronized and on the variance of only δ_{out}, the last link in the chain).

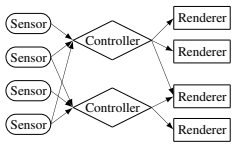


Figure 3: Generic architecture of an interactive multisensory virtual environment

4.2 Synchronization

Our architecture relies on accurate clock synchronization to compensate and circumvent timing problems. Fortunately, robust dis-

tributed clock synchronization is a well studied problem [27]. We will assume that the clocks on each node in our network have been synchronized as closely as desired. For most human interactions, clock synchronization to within a millisecond is sufficient and easy to achieve even on consumer operating systems. We used a simple multicast protocol to synchronize the clocks in our network.

4.3 Event Scheduling

Suppose now that we can forecast (perfectly), $\theta_{forecast}$ time units in advance, the time and location at which the event will occur. This means that we will be able to dispatch the event at time $t'_{predicted} = t_{detected} - \theta_{forecast}$ to the renderers, who will be able to initiate rendering at any $t'_{rendered} \geq t_{rendered} - \theta_{forecast}$. To correctly synchronize the output, we want $t'_{rendered} = t_{rendered} - \delta_{total}$; we will thus be satisfied provided $\theta_{forecast} \geq \delta_{total}$ (see Figure 2(c).

The above reasoning provides us with an estimate of how far into the future we must be able to forecast, but it leaves unresolved the question of exactly when to initiate rendering. The analysis in § 3 assumes that the latencies quoted are consistent; compensation using a fixed offset will result in the transmission of any latency variation through the the end-to-end perceived latency. In the naïve system, since the playback scheduling is "as soon as possible", the variability in every component of the latency contributes directly to the variability of the end-to-end latency. Analyzing the proposed architecture, it is important to reemphasize the two categories of latency; variation in $\delta_{circumvent}$ (provided the variation is bounded) will not be transmitted, but variation in δ_{in}, δ_{sense}, and δ_{out} will. Thus it is the temporal consistency of the sensors and renderers (not of data and event transmission) that is most important in maintaining the quality of the output in this architecture. Only the input and output latencies (δ_{in}, δ_{sense}, and δ_{out}) need to be compensated for when choosing the time at which to initiate rendering ($t'_{rendered}$). Since the prediction horizon builds in sufficient time to dispatch the event, a scheduled wait is introduced into the final phase of the architecture; this wait time acts as a buffer that absorbs the variation in $\delta_{circumvent}$.

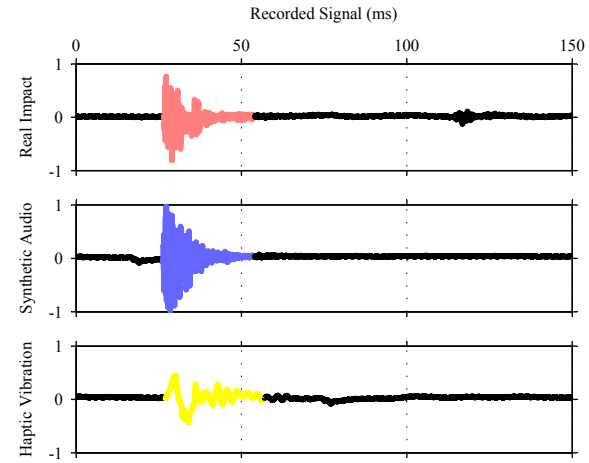
While we assumed above that we can forecast arbitrarily far into the future, our analysis establishes a *bound* for the minimum forecast horizon necessary to compensate for a given set of sensors and renderers and to circumvent the latency introduced by the application's logic.

5 Evaluation

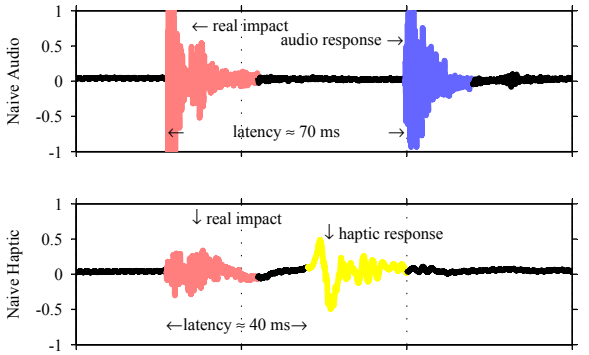
To test the feasibility of our spatio-temporal event architecture, we implemented a simple application that allows for precise measurement of the end-to-end latency of the entire sensing/prediction/rendering system.

The sensor input to our application consists of motion-tracking measurements of a probe's position. The measurements are obtained from a six-camera motion capture system [33] that tracks the location of optical markers on the probe and reconstructs the probe's position and orientation at approximately 120 Hz.

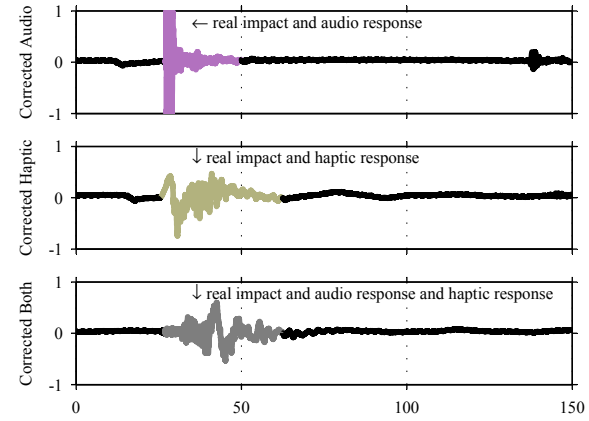
The event that this test detects/predicts is the impact of the tip of the probe with the surface onto which it is dropped from a height of about 40 cm. The system responds to the event by playing a collision sound over a loudspeaker and/or by rendering a force pulse on a haptic mouse [11]. Please see the accompanying video for an illustration of the environment setup. The test was implemented on a collection of 4 commodity PCs (ranging from a 2 GHz dual processor Xeon to a 3.2 GHz Pentium 4) connected by a gigabit network. The distributed components are Java applications running under Windows XP (except the sound renderer, which is a Java application running under Fedora Core 2 controlling an RME Hammerfall DSP).



(a) Isolated signal recordings



(b) Naïve solution latency



(c) Our solution

Figure 4: Audio recordings of the evaluation application. (a) shows the shape of the real impact sound as well as the sounds recorded from isolated playback of the audio and haptic responses. (b) shows typical examples of the latency between the real stimulus and the system's response under the naïve solution - note that the microphone placement for the haptic recording is such that the real impact sound is more muffled. (c) show the effect of using our event prediction model to reduce the latency between the real stimulus and the system's response - the two signals (three in the final plot) coincide.

For this application, the pertinent psychophysical thresholds are for auditory intramodal (2 ms) and audio-haptic intermodal (24 ms). To time the actual outputs, we exploit the fact that haptic devices can not avoid making a sound when they apply a force, and the probe also makes a sound when it hits the surface. By recording the output sounds from all devices (loudspeaker, haptic mouse, actual collision) with a single microphone, we can achieve accurate timing (better than 0.1ms) and perfect stimulus synchronization.

The recordings of the naïve solution (Figure 4 (b)) show significant delay (greater than the perceptual thresholds) between the onset of the real event and the system's response.

Observing the mean end-to-end latency observed in the naïve implementation, a simple kinetic predictor with a forecast horizon of 90 ms was implemented; tuning of the compensation offset yielded successful latency correction (Figure 4 (c) shows that the asynchrony between the real sound and the synthesized response is within the desired thresholds) with an 82 ms offset for the audio renderer and a 50 ms offset for the haptic renderer.

As well as showing the capability of the spatio-temporal event architecture to synchronize synthetic rendering with real phenomena, this evaluation application also reveals some of the boundaries of a system's effectiveness. Although the prediction architecture circumvents the internal latency, the variation in the compensated latency is transmitted (see § 4.3). This can be seen by examining the final two plots in Figure 4 (c); the location of the distinctive haptic onset varies within the ringing of the real impact sound, due largely to the maximum temporal resolution at which haptic playback can be initiated.

The sensor is the other aspect of this application's data chain that cannot be fully controlled. In the case of the Vicon sensor, the temporal variation arises from the preprocessing done by the motion tracker coupled with the fact that the data frames are sequentially numbered, rather than actually time-stamped.

6 Example Application

As well as our feasibility evaluation, we implemented an example AR application to illustrate how the spatio-temporal event prediction architecture can be scaled to meaningful application domains. Our application is a PONG-derivative table-top game incorporating multimodal user interaction. See Figure 1.

The playing surface and the virtual PONG ball are displayed on the table's surface using overhead video-projection. The user grasps and manipulates an iFeel haptic mouse, which is tracked by the Vicon motion tracking system. When a collision occurs with the virtual paddle represented by the mouse, the virtual ball is redirected, and a haptic pulse and a collision sound are rendered. Audio feedback is also used to perceptually reinforce ball bounces off the side walls and the ball dropping off the table when the player fails to intercept it.

To highlight the distributed flexibility of our architecture, the example application is structured with each system component running on a different commodity PC. Figure 5 illustrates the different components and the communication between them.

The Event Predictor uses the Vicon tracking information to determine when and where the user intercepts the PONG ball with the virtual paddle. In order to ensure that notification of changes to the ball's trajectory arrive at the renderers in time to allow coherent rendering, the interceptions must be predicted in advance. The time and location (and bounce angle) of the next interception is predicted by extrapolating the mouse's trajectory (assuming constant acceleration) from the most recent tracking information. If an interception is found within the forecast horizon (a tuneable parameter), an interception event is dispatched to the renderers. Since the renderers need to know in a timely manner about other events that affect the ball's trajectory (like wall bounces and miss-line crossings), if *no*

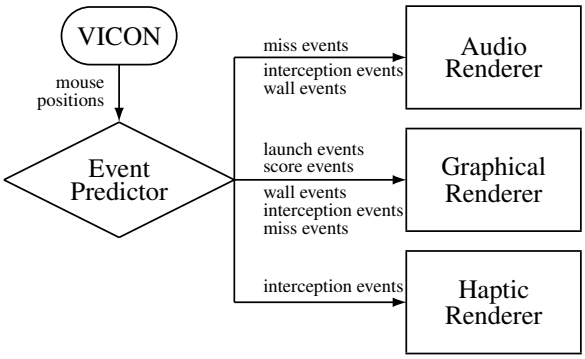


Figure 5: Distributed Structure

interception is found within the forecast horizon, but another system event will occur before the end of the forecast period, that event is dispatched to the renderers.

Three renderers are used in this application to create the user environment. The simplest is the Haptic Renderer that drives the iFeel's vibration motor in response to interception events. The Audio Renderer is similar to the Haptic Renderer in that it receives events relating to changes in the PONG ball's trajectory and schedules playback to contribute to the user's perception of impact (with different audio cues for paddle hits, wall bounces, and misses). The Audio Renderer's playback scheduling is more precisely controlled than in the Haptic Renderer; instead of waiting until the appointed time to begin playback, the Audio Renderer is constantly writing (silence) to the rendering hardware, and the response sounds are inserted into the written stream at the appropriate location to achieve the desired timing. The lower-level control provided by the hardware interface allows the application programmer to reduce the variation in δ_{out} for this modality (relative to the Haptic Renderer, where δ_{out} is subject to the timing vagaries of the iFeel API and USB communication).

The final output device is the Graphical Renderer, which uses structured-light reality augmentation to display the virtual PONG ball and the field of play on a blank table top. Since the unperturbed behaviour of the ball is deterministic, the Graphical Renderer only needs to be informed of the spatio-temporal events corresponding to changes in the ball's trajectory (launches, wall bounces, paddle hits, misses) or to other changes in the game state (e.g., score increases when the ball crosses the centre-line). The accompanying video shows the game in action.

This example is intended to show how our architecture is used in a meaningful (if simple) real-world application. The use of spatio-temporal event prediction allows for responsiveness in the interaction between real and virtual elements of the environment, and also facilitates the synchronization of the different modalities (visual, auditory, and haptic). By ensuring that the forecast horizon is sufficiently large to allow in-time rendering by the different renderers, the different modalities combine to reinforce the perception of seamlessness between real and virtual.

7 Conclusions

We described a simple, yet surprisingly effective, architecture for distributed rendering of multisensory events. Based on our experiments, we make the following observations.

• Separation of latency from synchronization is easy in our architecture, and significantly reduces asynchrony. For further improvements, it is sufficient to focus on the renderer latency and clock synchronization.

• The effect of latency variation on synchronization makes accurate time-stamping of sensor data a worthwhile area of focus.

For example, in using Vicon's real-time output as our sensor, we observed that Vicon's "sensor readings" include temporal filtering, and can arrive at variable rate, but are stamped with a frame-number which is not consistently correlated with the time a given state occurred.

• Although not all AR applications can be structured in a purely event-based architecture like we have described (e.g., those where tracking is tightly coupled with output on a frame-by-frame basis), hybrid architectures could be implemented that combine an event-based structure with additional data-paths where necessary for continuous phenomena.

• The calibration of the forecast horizon and the latency compensation offset should be performed sequentially in two phases. The presence of a sufficient forecast horizon allows the pertinent compensatable latencies to be isolated.

• Simple predictive models are often sufficient to achieve the necessary forecast horizon. In our applications, simple 1-D and 2-D kinetic predictors were sufficient to obtain adequate results. While domain knowledge of the mechanics of an interaction could facilitate more sophisticated prediction models, it is worth noting the effectiveness of easy-to-implement solutions.

Acknowledgements: This material is based on work supported in part by NSF grants IIS-0308157, EIA-0215887, ACI-0205671, EIA-0321057, NIH(CRCNS) grant R01 NS50942, and SimQuest LLC.

REFERENCES

[1] Bernard D. Adelstein, Durand R. Begault, Mark R. Anderson, and Elizabeth M. Wenzel. Sensitivity to haptic-audio asynchrony. In *Proceedings, 5th International Conference on Multimodal Interfaces*, pages 73–76, 2003.

[2] Ronald Azuma. A survey of augmented reality. *Presence*, 6(4), 1997.

[3] Ronald Azuma and Gary Bishop. Improving static and dynamic registration in an optical see-through hmd. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 197–204, New York, NY, USA, 1994.

[4] Roland Blach, Jürgen Landauer, Angela Rösch, and Andreas Simon. A highly flexible virtual reality system. *Future Gener. Comput. Syst.*, 14(3-4):167–178, 1998.

[5] P. Buttolo, R. Oboe, and B. Hannaford. Architectures for shared haptic virtual environments. *Computers and Graphics*, 21:421–429, 1997.

[6] Derek DiFilippo and Dinesh K. Pai. The AHI: an audio and haptic interface for contact interactions. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 149–158, New York, NY, USA, 2000. ACM Press.

[7] N. F. Dixon and L. Spitz. The detection of auditory visual desynchrony. *Perception*, 9, 1980.

[8] Emmanuel Frécon and Mårten Stenius. DIVE: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal*, 5(3):91–100, September 1998.

[9] Aaron Garrett. Reduction of latency in virtual environments. Master's thesis, Jacksonville State University, 2002.

[10] Aaron Garrett, Mario Aguilar, and Yair Barniv. A recurrent neural network approach to virtual environment latency reduction. In *IJCNN'02. Proceedings of the 2002 International Joint Conference on Neural Networks*, volume 3, pages 2288–2292, 2002.

[11] iFeel. `http://www.immersion.com/`.

[12] Marco C. Jacobs, Mark A. Livingston, and Andrei State. Managing latency in complex augmented reality systems. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 49–54, New York, NY, USA, 1997. ACM Press.

[13] J. Y. Jung, B. D. Adelstein, and S. R. Ellis. Predictive compensator optimization for head tracking lag in virtual environments. In *IMAGE (Innovative Modeling and Advanced Generation of Environments)*, pages 123–132, 2000.

[14] D. J. Levitin, M. V. Mathews, and Karon MacLean. The perception of cross-modal simultaneity. In *International Journal of Computing Anticipatory Systems*, 1999.

[15] Jiandong Liang, Chris Shaw, and Mark Green. On temporal-spatial realism in the virtual reality environment. In *UIST '91: Proceedings of the 4th annual ACM symposium on User interface software and technology*, pages 19–25, New York, NY, USA, 1991. ACM Press.

[16] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A network software architecture for large-scale virtual environments. *Presence*, 3(4):265–287, 1994.

[17] Michael R. Macedonia and Michael J. Zyda. A taxonomy for networked virtual environments. *IEEE MultiMedia*, 4(1):48–56, – 1997.

[18] Katerina Mania, Bernard D. Adelstein, Stephen R. Ellis, and Michael I. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pages 39–47, 2004.

[19] João Luís Marins, Xiaoping Yun, Eric R. Bachmann, Robert B. McGhee, and Michael J. Zyda. An extended kalman filter for quaternion-based orientation estimation using MARG sensors. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2003–2011, 2001.

[20] Martin Mauve. Consistency in replicated continuous interactive media. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 181–190, 2000.

[21] Martin Mauve, Jürgen Vogel, Volker Hilt, and Wolfgang Effelsberg. Local-lag and timewarp: Providing consistency for replicated continuous applications. 6(1):47–57, February 2004.

[22] Mark R. Mine. Characterization of end-to-end delays in head-mounted display systems. Technical Report TR93-001, University of North Carolina at Chapel Hill, 1993.

[23] Nadine E. Miner and Thomas P. Caudell. Computational requirements and synchronization issues for virtual acoustic displays. *Presence*, 7(4):396–409, 1998.

[24] Dinesh K. Pai. Multisensory interaction: Real and virtual. In *Robotics Research: The Eleventh International Symposium*, volume 15. 2005.

[25] Steve Pettifer, Jon Cook, James Marsh, and Adrian West. DEVA3: architecture for a large-scale distributed virtual reality system. In *VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 33–40, New York, NY, USA, 2000. ACM Press.

[26] Steve Pettifer and Adrian West. Subjectivity and the relaxing of synchronization in networked virtual environments. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 170–171, New York, NY, USA, 1999. ACM Press.

[27] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, 1990.

[28] Matthew J. P. Regan, Gavin S. P. Miller, Steven M. Rubin, and Chris Kogelnik. A real-time low-latency hardware light-field renderer. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 287–290, 1999.

[29] Xiaojun Shen, Jilin Zhou, Abdulmotaleb El Saddik, and Nicolas D. Georganas. Architecture and evaluation of tele-haptic environments. In *DS-RT '04: Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 53–60, 2004.

[30] Shervin Shirmohammadi and Nicolas D. Georganas. An end-to-end communication architecture for collaborative virtual environments. *Comput. Networks*, 35(2-3):351–367, 2001.

[31] Kay M. Stanney, Ronald R. Mourant, and Robert S. Kennedy. Human factors issues in virtual environments: A review of the literature. *Presence*, 7(4):327–351, 1998.

[32] Russell M. Taylor II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. VRPN: a device-independent, network-transparent vr peripheral system. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61, New York, NY, USA, 2001. ACM Press.

[33] Vicon. `http://www.vicon.com`.

[34] Ingrid M. L. C. Vogels. Detection of temporal delays in visual-haptic interfaces. *Human Factors: The Journal of the Human Factors Society*, 46(1):118, 2004.

[35] M. Wloka. Lag in multiprocessor virtual reality. *Presence*, 4(1):50–63, 1995.