On Error Bound Estimation for Motion Prediction

Rynson W.H. Lau Kenneth Lee

Department of Computer Science, City University of Hong Kong, Hong Kong

ABSTRACT

A collaborative virtual environment (CVE) allows remote users to access and modify shared data through networks, such as the Internet. However, when the users are connected via the Internet, the network latency problem may become significant and affect the performance of user interactions. Existing works to address the network latency problem mainly focus on developing motion prediction methods that appear statistically accurate for certain applications. However, it is often not known how reliable they are in a CVE. In this work, we study the sources of error introduced by a motion predictor and propose to address the errors by estimating the error bounds of each prediction made by the motion predictor. Without loss of generality, we discuss how we may estimate the upper and lower error bounds based on a particular motion predictor. Finally, we evaluate the effectiveness of our method extensively through a number of experiments and show the effectiveness of using the estimated error bound in an area-based visibility culling algorithm for DVE navigation.

KEYWORDS: collaborative virtual environments, network latency, motion prediction, prediction error.

1 INTRODUCTION

A collaborative virtual environment (CVE) allows remote users to access and modify shared data through networks, such as the Internet. Example applications of CVEs include collaborative design and online gaming. However, when the users are connected via the Internet, the network latency problem may become significant and may seriously affect the performance of user interactions. According to our experiments, a typical cross country single-trip delay is between 80ms to 180ms.

In a CVE, a user (referred to as **A**) needed to interact with an object typically does it based on the current state of the object. However, if a remote user (referred to as **B**) is also interacting with the same object at the same time, the changes to the object due to **B** will not be available to **A** until after the network delay. In other words, while two users are editing an object concurrently, the object state presented to one user may be different from that presented to the other user. This problem is generally referred to as the consistency problem.

A popular solution to address the consistency problem is motion synchronization. Recently, a motion synchronization method based on minimizing the motion discrepancy between the client and the server was proposed [13]. Although this method is shown to be effective in reducing motion discrepancy, like most motion synchronization methods, it relies on a motion predictor to compensate for the network latency. For example, if client **A** has just received a location update message from client **B** with a known network delay of 0.2s, it means that **B** was at the given location 0.2s ago. A typical synchronization method will then use a motion predictor to predict the current location of **B** based on **B**'s location 0.2s ago. With this predicted position, the motion synchronization method may then try to minimize the discrepancy between clients **A** and **B** by moving the avatar of **B** in client **A** to the predicted position. Here, the accuracy of the motion predictor plays a very important role in the effectiveness of the motion synchronization method. Unfortunately, most existing works consider only the statistical accuracy of motion predictors. To our knowledge, no work has been published on estimating the error bounds of each prediction made.

The ability to predict the error bound, i.e., to set a lower and an upper bounds as shown in Figure 1, for each prediction made allows us to estimate the reliability of individual predictions. This has many important applications to CVEs. For example, assume that client **B** is modifying an object. If client **A** is also interested in the same object, we may superimpose the object updated with the lower error bound location and that with the upper error bound location onto the object updated with the predicted location. This will provide **A** some idea of the scale of the consistency problem and allow **A** to decide if he/she wants to edit the object concurrently. In case of a critical application, we may even temporary disallow **A** to edit the object concurrently if the estimated error bounds are too wide apart. In addition, in some applications where geometry data are dynamically streamed to the clients (e.g., very large dataset [7] and data generated from many remote sources [1]), these data need to be prefetched to the clients before they are accessed. If we know that a particular prediction has a small error, we may adopt an optimistic approach in prefetching to save network bandwidth by not streaming geometry data which are relatively far from the predicted position. Further, in large-scale DVEs, the estimated error bounds allow us to determine a region, instead of a single point, where the user may be at a particular future moment. We may use the predicted region to compute the potential visible set (PVS) with a region-based visibility culling algorithm [8].

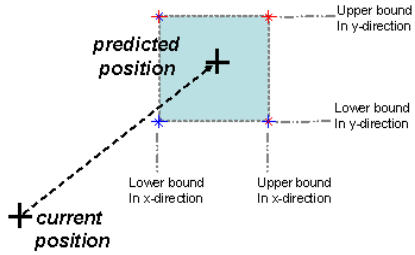

Figure 1. Predicting the upper and lower error bounds of a prediction.

In this paper, we analyze the sources of prediction error in motion prediction. To study how these sources affect the prediction accuracy, we have used the motion prediction method proposed by [4, 5] to investigate how these sources of error affect the prediction accuracy and how we predict the upper and lower error bounds. Although our discussion is based on a single predictor, the ideas discussed here are expected to be easily applied to other motion predictors.

The rest of the paper is organized as follows. Section 2 briefly summarizes related work on motion prediction and on error estimation. Section 3 studies the sources of error in motion

IEEE Virtual Reality 2010
20 - 24 March, Waltham, Massachusetts, USA
978-1-4244-6236-0/10/$26.00 ©2010 IEEE

prediction. Section 4 analyzes the characteristics of a motion predictor and how to reduce the effects of the sources of prediction error in order to be able to estimate a more reliable error bound. Section 5 investigates the reliability of the estimated error bounds through a number of experiments. Finally, Section 6 briefly concludes the work presented in this paper.

2 Related Work

There has been a lot of work on motion prediction. However, there is not much work on estimating the error and the reliability of the prediction. In this section, we briefly summarize existing work on motion prediction and on error estimation.

2.1 Motion Prediction

A popular solution to address the network latency problem is dead reckoning [15]. For each shared object, a motion predictor is run on the host, i.e., the machine that has control on the motion of the object, as well as on each of the clients accessing the object. The host only sends out an update message when the error between the predicted state and the actual state of the object is higher than a given threshold. In between updates, the clients approximate the object state with the local motion predictor. In general, dead reckoning helps reduce not only the network latency problem, but also the number of update messages needed to be sent through the network. However, the performance of the dead reckoning scheme depends heavily on the accuracy of the predictor used. Among all, the polynomial predictors are most popularly used with the dead reckoning scheme to help extrapolate/predict future locations of objects. They are general predictors and are not designed for any specific objects. In [7], an EWMA scheme was used for object prefetching in a DVE. It assigns different weights to past movement vectors, with higher weights to recent ones.

There has been some work on predicting human body motion. In [2], a predictor for head motion was proposed. It is based on the Kalman filter, which was originally used to filter measurement noise in linear systems by recursively minimizing the mean square prediction error. This method may work with a prediction model to further reduce the prediction error. Results show that during rapid motion, the predictor becomes less effective and its performance is similar to one without using the Kalman filter. Another method is to use the Gauss-Markov process model to predict head motion [14] and human motion [3]. In [18], a grey system theory-based predictor, which accuracy is similar to the polynomial-based predictor with Kalman filtering, is used to predict head motion. Other than Kalman filtering, sequential Monte Carlo [11] is used in motion tracking. Although this method can approximate arbitrary distribution functions without uni-model Gaussian assumption, it is relatively expensive computationally. In [12], a double exponential smoothing-based predictor is proposed. It is efficient despite its slightly lower accuracy. In [4], a mouse-based motion predictor was proposed, which considers the bio-kinematic constraints of the human arm in the prediction. This work was later extended to model finger motion to support hand interactions [6].

2.2 Error Estimation

Prediction errors have been mainly studied in the context of Kalman filters [16]. In Kalman filtering, whenever a measurement is received, the residual between the measurement and the predicted value is computed. This residual is then used to adjust the Kalman filter to generate the next prediction so as to minimize the residual of the next prediction. However, this technique is, straightly speaking, not an error prediction method, as it simply uses the actual error from the last prediction to adjust the next prediction. In [10], two sources of prediction error were considered: model inaccuracy and input noise. It characterizes these two sources of prediction error statistically, in the form of an open-loop Kalman filter. However, this work does not estimate nor quantify the magnitude of prediction errors. [9] studied the prediction error of polynomial models. It is more a theoretical analysis, without considering the effect of network latency on prediction errors.

Compared to these works, our work can be considered as more aggressive in that we attempt to estimate a lower error bound and an upper error bound of each prediction by estimating the error of the prediction. While the prediction tells us the most likely position of the tracked object at a particular future moment, the error bounds help us determine the region, within which the tracked object will most likely be at that future moment.

3 General Analysis of Prediction Errors

In general, there are two ways to drive an object to move. One is using a program, in which the object motion is often clearly defined. This type of objects is typically referred to as autonomous objects. Another way is by some external factors. A popular type of objects is referred to as avatars, which are virtual characters representing the real users and their motions are driven by the users interactively through some input devices, e.g., mice or joysticks. Typically, they are the objects which motions need to be predicted and they are also the focus of this paper.

To model the behavior of prediction errors, the understanding of error sources is essential. We have identified four sources of prediction error here. This section provides a non-method-specific overview of these four sources of error. In the following discussions, we refer to the actual motion behavior of the object as *correct motion model* (f_c) and the motion model used to model the object motion as the *approximated motion model* (f_a).

Mismatch of Motion Model: This error is caused by the use of an incorrect model to model the motion behavior of objects. For example, the second order polynomial (SOP) predictor, which is based on the Newton's Law of Motion, is a very popular method used to model general object motion. However, a lot of objects do not actually follow this motion model. In general, the prediction error due to the use of an incorrect motion model is affected by:

- The deviation between the approximated motion model and the correct motion model.
- The *prediction length*, i.e., how far ahead in time that we want to predict.

and can be written as:

$$E_1 = f_a(s, t_s + t_l) - f_c(s, t_s + t_l) \quad (1)$$

where t_s is the sampling time (relative to the start of the motion) of the last received update, s is the internal state of the predictor (e.g., position, velocity, or acceleration) obtained at t_s, and t_l is the prediction length. As shown in Figure 2, the difference between the approximated motion model and the correct motion model at the prediction time, i.e., $t_s + t_l$, is the prediction error.

To address this error, an ideal solution is to determine the correct motion model that determines the object motion. However, this is often not possible to do when the motion is driven by an external force. Even if we can determine a suitable motion model, it may still require many motion parameters as input in order for the motion model to be accurate and it is likely that either not all of these parameter values are available or it may be computationally too expensive to consider all of them. A typical solution to this problem is to continuously refine the approximated motion model with the input measurements. However, the prediction accuracy of this solution is still affected by the prediction length.

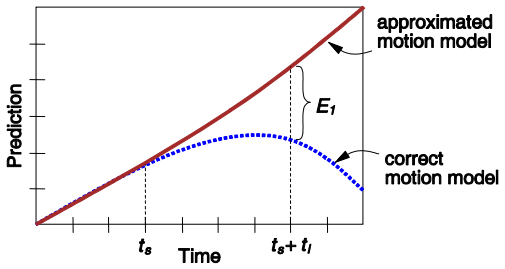

Figure 2. Prediction error due to mismatch of motion model.

Input Noise: This error is caused by the deviation of the input measurements from the motion behavior of the input device, due to some random factors such as surface friction, mechanical faults and some sudden involuntary movements. The deviation in input measurements will affect the internal states of the predictor and hence the prediction. In Kalman filtering, this noise is typically assumed to be white Gaussian, i.e., a Gaussian distribution with a zero mean. The input noise normally has a much higher effect at the beginning of motion while the motion model is being constructed, as the measurement values at this moment are typically low and the number of measurements obtained is small. Once the motion model is established and refined, the effect of input noise will be significantly reduced. The prediction error due to input noise, E_2, can be written as follows:

$$E_2 = f_a(s + \Delta n \frac{\partial s}{\partial p}, t_s + t_l) - f_a(s, t_s + t_l) \quad (2)$$

where $\partial s/\partial p$ is the change in the state due to a small change in input measurement, while Δn is the amount of input noise. However, it is not possible to determine the actual noise value from an input measurement. In Kalman filtering, the designer of the filter needs to know the noise level of the input device and to indicate in advance how much the predictor should trust the predicted values and the input measurements. We have conducted some experiments. A simple low pass filtering operation can significantly reduce the effect of noise on individual measurements, at the cost of increasing sampling delay in particular when the sampling frequency of the input device is low.

Quantization Error: This error is caused by the quantization of measurement values. Most input devices return integer position values. If the resolution of an input device is low, the quantization error becomes high. The prediction error due to quantization can be written as:

$$E_3 = f_a(s + \Delta p \frac{\partial s}{\partial p}, t_s + t_l) - f_a(s, t_s + t_l) \quad (3)$$

where Δp is the amount of input quantization. Figure 3 shows the effect of quantization on the accuracy of the motion model.

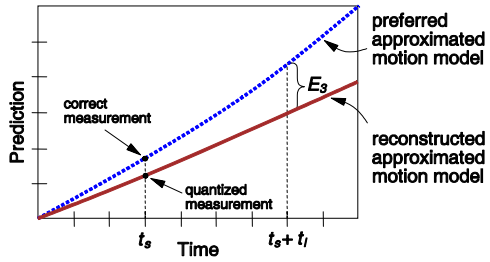

Figure 3. Prediction error due to input quantization.

Although the amount of quantization may be small, it is magnified by the prediction length. Like input noise, quantization normally has a higher effect at the beginning of the motion while the motion model is being constructed, due to the low measurement values and the small number of measurements received. Once the motion model is established and refined, the effect of quantization will be significantly reduced. Again, we may apply a low pass filter on the input measurements to reduce the effect of quantization.

Sampling Time Precision: This error is caused by the uncertainty of when exactly each input measurement is taken. Although we may set the frequency for the device driver to sample the input device, it is usually not known when exactly the device driver samples the input device. In addition, as the time that the device driver samples the input device is not synchronized with the time that the device application reads the sampled value, it is often not known which sample that the device application gets from the device driver each time.

Figure 4 shows this time precision problem. The device driver reads the input measurement at time t_a, which corresponds to the preferred approximated motion model. However, the device application obtains the measurement at t_s, which causes the constructed approximated motion model to deviate from the preferred model. E_4 is the prediction error due to the use of the constructed model with prediction length t_l as follows:

$$E_4 = f_a(s + \Delta t \frac{\partial s}{\partial t}, t_s + t_l) - f_a(s, t_s + t_l) \quad (4)$$

where $\partial s/\partial t$ is the change in the state due to a small change in the sampling time, while $\Delta t = t_a - t_s$ is the actual deviation in sampling time. Although the amount of deviation may be small, it is magnified by the prediction length.

To address this problem, in case if it is possible to modify the device driver, we may rewrite it such that every time it samples the input device, it immediately checks the local timer and returns the input measurement together with the timer value. This timer value provides a very close approximation of the sampling time of the measurement. However, if modifying the device driver is not possible, we may modify the device application to check the local timer every time it reads a sampled value from the device driver. Unfortunately, this timer value is not as accurate as the one obtained by the device driver.

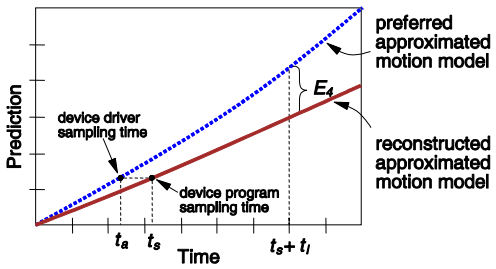

Figure 4. Prediction error due to the uncertainty in sampling time.

4 Error Analysis of a Mouse-Based Motion Predictor

In this section, we look at how the four sources of error discussed in the last section affect prediction accuracy. Our discussion will be based on a mouse-based motion predictor proposed by [4, 5]. Note that we have chosen this predictor for this work because the method is robust. As pointed out in [6], most of the devices directly manipulated by human beings are also affected by similar motion constraints and hence the mouse-based motion prediction method could also be applied to those devices too. The paper demonstrated this point by extending the mouse-based method for predicting hand gesture.

In the following subsections, we first briefly summarize this mouse-based motion predictor and then discuss the various sources of prediction errors in such a predictor.

4.1 A Mouse-Based Motion Predictor

Through studying the motion behavior of a mouse during 3D navigation, [4] notes that a typical navigation step contains a number of motion pulses as shown in Figure 5. Each pulse may represent a forward or backward movement and can be approximated by a Gaussian curve. The paper then proposes an elliptic model to model the motion behavior of the 2D mouse during 3D navigation. The elliptic model is defined as follows:

$$v = K_1(\cos(K_2\ t) - 1) \quad (5)$$

where v is the predicted velocity at the prediction time t, since the start of the motion pulse. K_1 and K_2 are referred to as the pulse constants. They are the parameters of the motion pulse and are constants during the period of the pulse. While the magnitude of K_1 is proportional to the height of the pulse, K_2 is inversely proportional to the width of the pulse.

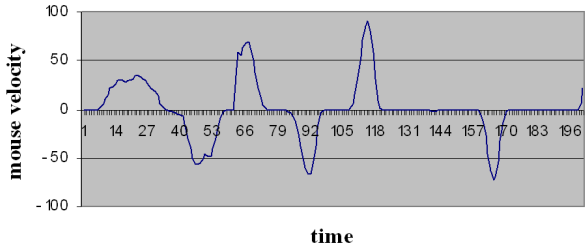

Figure 5. Mouse motion velocity of a sample navigation step.

The predicted mouse position is computed as follows [5]:

$$p_{new} = \frac{K_1}{K_2}\left(\sin(K_2(t_s + t_l)) - \sin(K_2 t_s)\right) - K_1 t_l + p \quad (6)$$

where t_s is the current time. t_l is the prediction length. p and p_{new} are the current and predicted mouse positions, respectively. The 2D mouse motion data on the screen space is divided into two time-series, x and y motions. Two separate motion predictors are used to predict the mouse motion velocities in x and y directions. By mapping the predicted 2D velocity vector to 3D space, the future position of the avatar in the 3D scene can be predicted.

4.2 Error Analysis

In Section 3, we have investigated the sources of error in motion prediction. Since the four sources of error have different effects on different parts of a motion pulse, we divide a motion pulse into three stages to investigate the prediction accuracy of the predictor. We discuss how the four sources of error affect the three stages of the motion pulse and how we may determine the upper and lower error bounds of each prediction.

As pointed out by [6], most devices directly manipulated by human beings have a velocity motion pulse similar to a Gaussian pulse. The device's velocity starts from zero at the beginning and gradually increases until reaching a maximum value. It then drops until it returns to zero, restricted by the finite spatial coverage of the human body part that manipulates the device. The paper suggests that the elliptic model used by the mouse-based motion predictor is a good match to most of the devices directly manipulated by the user in CVEs. Apart from mice and gloves, it can also be applied to predict motion of other devices such as driving wheels, Nintendo Wii, etc.. Since we have used a mouse for interaction in all our experiments conducted in this work, the use of the elliptic model should address the first source of error, mismatch of motion model. Once again, our analysis here should be easily applied to other devices that are directly manipulated by human beings.

To study the prediction error, we capture a short sequence of the x-axis mouse motion as shown in Figure 6. The motion pulse represents a single forward movement of the mouse. It can be divided into three stages: **I**. Pulse start, **II**. Model refinement, and **III**. Pulse end. Figure 7 provides some idea of the prediction error. It is computed as:

$$\text{Prediction Error (\%)} = \left| \frac{\text{predicted position - actual position}}{\text{total displacement of the pulse}} \right| \quad (7)$$

To obtain Figure 7, a prediction step, i.e., the basic reference unit, is set at 50ms. We can see that when the prediction length, i.e., the duration ahead in time that we want to predict, is low (50ms), the prediction error is relatively small. However, when the prediction length is set to 5 prediction steps (250ms), the prediction error is much higher.

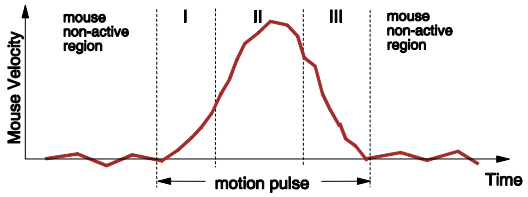

Figure 6. A typical motion pulse.

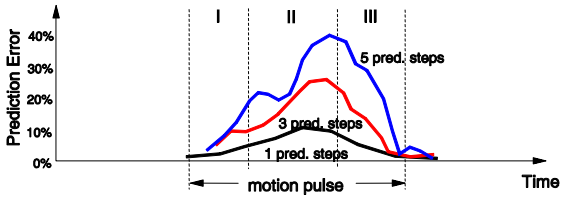

Figure 7. Prediction error vs. prediction length (1,3,5 prediction steps).

During stage **I**, the mouse suddenly changes from non-active stage into an active stage, while the client does not know it yet due to network latency. This is the most difficult stage in motion prediction as there is no way we can predict when this sudden change of motion will happen. This is a fundamental limitation of motion prediction. As we are not able to perform any prediction at this stage, we propose to bound the error here. In general, there are three ways to do this. We can bound the position, the velocity or the acceleration. However, bounding the position, i.e., distance, would introduce motion jerkiness (sudden stopping of motion). Bounding the velocity, on the other hand, would result in non-uniform motion. Hence, we suggest to bound the acceleration here. When the mouse acceleration exceeds a maximum threshold, a_{max}, it will be clipped to a_{max}. However, in order to minimize the effect of this clipping on the internal states of the Kalman Filter, we only apply this clipping on the first few measurements. Hence, we set the upper error bound (B_u) and the lower error bound (B_l) as follows:

B_l = latest received measurement value

$B_u = a_{max}\ t_l^2$

During this stage, the remote machine will have received two to three input measurements from the host machine. Based on the received measurements, it detects if they represent the start of a new motion pulse and constructs the motion model for the pulse. Due to the limited number of measurement values, the constructed motion model may not be very accurate.

During stage **II**, the mouse motion is expected to accelerate. The accuracy problem of the constructed model is escalated by the accelerating motion. In addition, a long prediction length would significantly magnify the prediction error as shown in Figure 7. In general, the prediction accuracy at this stage is affected by the quantization, noise and time precision problems. To address these problems, we timestamp each input measurement as we construct the approximated motion model in stage **I**. During stage **II**, we also try to refine the approximated motion model as we receive more measurements from the host machine.

To estimate the error bounds, we consider two situations – when the filter has nearly converged and when it is not. The upper bound can be determined as follows:

$$B_u = p + \left[\mathrm{Max}\left(f_a(s^-, t_s + t_l), \max_{i=-\delta}^{+\delta} \left(f_a(s + iKt_l\Delta s, t_s + t_l) \right) - p \right] F \quad (8)$$

where s^- is the *a prior* state variable (the last pulse constants). p is the latest received measurement. Δs is an arbitrary small constant to jitter the internal state of the Kalman Filter. δ determines the amount of jitter. F is a scaling factor between 1 and 2, and will be discussed next. $f_a()$ is the reconstructed approximated motion model. The inner Max() determines the maximum predicted position due to the jittering of the internal state, when the filter has not converged. The $f_a()$ on the left computes the predicted position using the last internal state, when the filter has nearly converged. This is because if the filter is close to convergence, the last and the next internal states should be similar.

Based on these two computed positions, we select the maximum one to form the upper error bound. Note that in our implementation, Δs is set to 0.5 and δ to 4. Once we have determined the upper bound position, we flip it about the predicted position to become the lower bound position as follows:

$$B_l = \mathrm{Max}\left(p, f_a(s, t_s + t_l) - \left(B_u - f_a(s, t_s + t_l)\right)\right) \quad (9)$$

The right term of Max() is to compute the flipped position. Max() is to bound the computed value so that the lower bound position would not go behind the last received measurement p.

The scaling factor F is to scale the upper and lower bounds. It is computed in proportional to the area within the relevant time period, i.e., between t_s and $t_s + t_l$, of the motion model as:

$$F = \mathrm{Max}\left(1, \frac{1}{t_l} \sum_{t=t_s}^{t_s+t_l} \left(1 - \cos(K_2 t)\right)\right) \quad (10)$$

Here, the summation term accumulates the predicted mouse velocity within the time period. After averaging the sum by the prediction length, we obtain a scaling factor between 0 to 2. Max() is to bound the value of F between 1 and 2.

During stage **III**, the mouse motion velocity gradually reduces back to the non-active stage. As the client has already received enough input measurements by then, the motion model has been refined into a rather accurate form. Hence, during this phase, the mouse motion is typically more stable and predictable, and the prediction error is generally low. As such, we may use a tighter error bound. Here, we may compute the upper and lower error bounds in the same way as in stage **II**. In Eq. (10), we define the scaling factor F as a function of the angular value. When t is close to π/K_2, i.e., near to the middle of the motion pulse, F is at its maximum value and hence, the upper and lower bounds become further apart. When t is close to $2\pi/K_2$, i.e., near to the end of the motion pulse, F is at its minimum value and hence, the upper and lower bounds become closer. So, without any change, we may use the same error bounds here.

5 Experimental Results and Discussions

In this section, we show some experiments that we have conducted to investigate the reliability of the error bounds based on the mouse-based motion predictor. A prediction step is set at 50ms. This is the basic unit that the application uses to read the mouse actual positions (input measurements) and to send out update messages from the host to the client. To evaluate our method, we have used two prediction lengths in the experiments: 1 prediction step, i.e., 50ms, and 5 prediction steps, i.e., 250ms. We present a total of five different experiments here.

5.1 Experiment 1

Our first experiment is to show the effect of the acceleration clipping on prediction error. Here, we only apply acceleration clipping on the first few samples of each motion pulse to minimize its effect on the rest of the pulse. In Figure 8, we can see that after clipping the acceleration of the mouse motion, the total mouse displacement is slightly reduced. We can also see that the predicted position and the actual position are very close most of the time, due to the short prediction length of 1 prediction step. Figure 9 increases the prediction length to 5 prediction steps. We can see that the predicted curves (with and without acceleration clipping) fluctuate more significantly here. This is mainly due to the higher prediction length. We can also see that at the 8th sample point, the displacement of the predicted curve with acceleration clipping is smaller than the one without clipping. This is because the mouse acceleration at this point exceeds the threshold and is clipped. Note that the total duration, i.e., the length of the horizontal axes of Figures 8 and 9, is equal to the pulse length, which varies depending on individual pulses.

From our experiments, we have found that the reduction in prediction error with acceleration clipping is very small, about 1%. This is mainly due to the fact that most of the motion pulses have low acceleration at the beginning of the pulse.

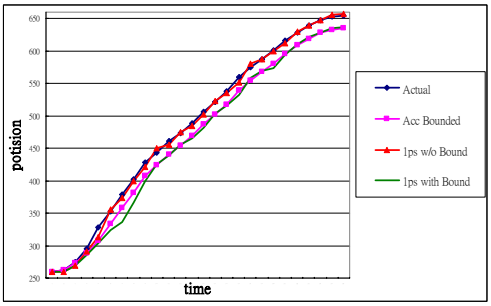

Figure 8. Acceleration clipping at prediction length = 1 prediction step.

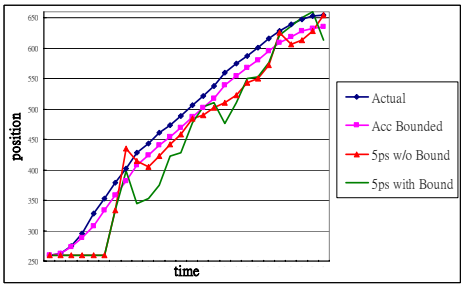

Figure 9. Acceleration clipping at prediction length = 5 prediction steps.

5.2 Experiment 2

In this experiment, we show the effect of time-stamping the measurements and readjusting the values on prediction error. To

perform this experiment, we deliberately jitter each measurement by a mean of 7.5ms or a maximum of 15ms (since the default sampling time of the mouse device driver is set to roughly 15ms). In Figure 10, we can see that there is a small difference between the actual curve (original measurements) and the jittered curve (with jittered measurements), but the predicted curve with time jittering deviates from that without jittering by a considerable amount, when the prediction length is set to 1 prediction step. However, if we increase the prediction length to 5 prediction steps as shown in Figure 11, the predicted curve with time jittering fluctuates, and hence deviates from that without jittering, even more significantly as a result of the long prediction length. From our experiments, we have also found that the increase in prediction error due to time jittering is 0.2% when the prediction length is set at 1 prediction step. However, if we increase the prediction length to 5 prediction steps, the prediction error is increased to 7%.

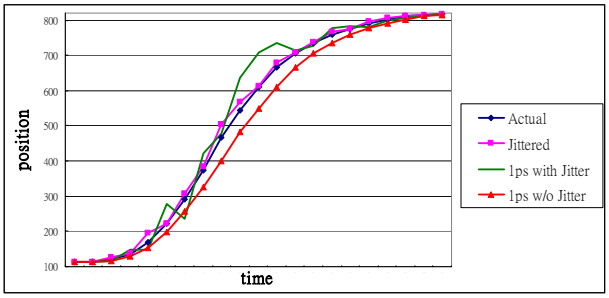

Figure 10. Time-stamping at prediction length = 1 prediction step.

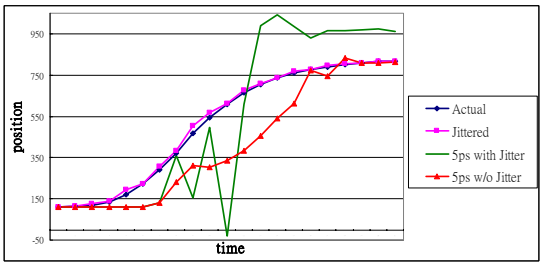

Figure 11. Time-stamping at prediction length = 5 prediction steps.

5.3 Experiment 3

In this experiment, we show the error bounds computed with our method. In Figure 12, we show the error bounds when the prediction length is set to 1 prediction step. We can see that the computed error bounds are formed on the two sides of the predicted curve and are rather tight when the prediction length is short. We can also see that most of the measurements on the actual curve fall inside the bounds, except only 1 measurement near to the middle that falls outside the upper error bound.

Figure 13 shows the error bounds computed when the prediction length is set to 5 prediction steps. We can see that the computed error bounds now become much wider apart. Due to the fluctuation of the predicted curve, the upper and the lower error bound curves also fluctuate quite significantly. We can also see that there are now a few more measurements that fall outside of the upper bound. (In fact, the measurements may fall outside of the lower bound in some situations, although this happens less frequently.) This shows that occasionally, the error bounds may fail to bound the error. In other words, our approach to compute the error bounds may not guarantee that all the measurements will fall inside the error bound. There is in fact a tradeoff here. As we widen the error bounds to cover all the measurements by changing the scaling factor *F*, we also reduce the tightness of the bounds. When these bounds are used for dataset prefetching, the amount of data needed to be prefetched increases quadratically as we widen the distance between the upper and the lower bounds. In fact, for some motion pulses which are less regular, widening the error bounds to cover all the measurements may not be effective, as the predicted curve may fluctuate very significantly.

Note that although we compute the lower error bound by flipping the upper error bound about the predicted position, we can see from both figures that the upper bound curves are not symmetrical to the lower bound curves about the predicted curve. This is because the lower error bound curve is bounded by the received input measurement to prevent the predicted curve from moving behind the actual motion. For example, at the 13rd sample point of Figure 13, the lower error bound is constrained by the actual input measurement at the 8th sample point.

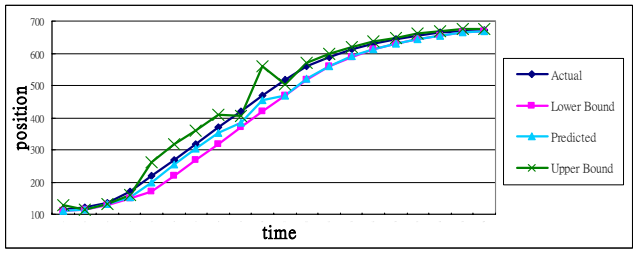

Figure 12. The estimated error bounds of a single motion pulse at prediction length = 1 prediction step.

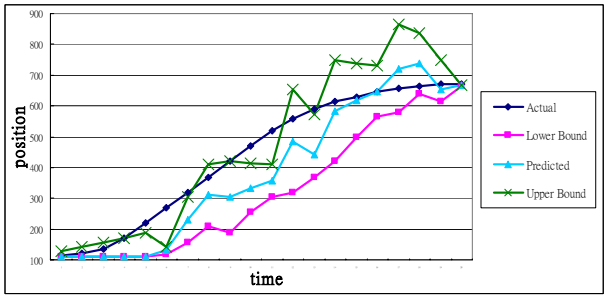

Figure 13. The estimated error bounds of a single motion pulse at prediction length = 5 prediction steps.

5.4 Experiment 4

To study the reliability of the estimated error bounds, we have collected over 100 motion pulses and computed the statistical results of all the sample points within each of the three stages of the motion pulses. Table 1 shows statistical results including the deviations of the error bounds from the actual positions (input measurements), and the tightness of the computed error bounds. Note that we use a high prediction length of 5 prediction steps, i.e., 250ms, here to study the reliability of the error bounds. The definitions of the terms are defined as follows:

$$\text{Upper Deviation} = \left| \frac{\text{upper bound - actual position}}{\text{total displacement of the pulse}} \right|$$

$$\text{Lower Deviation} = \left| \frac{\text{lower bound - actual position}}{\text{total displacement of the pulse}} \right|$$

$$\text{Bound Tightness} = \frac{\text{upper bound - lower bound}}{\text{total displacement of the pulse}}$$

From Table 1, we can see that there is a tighter average error bound at stage **III** due to the fact that we have a more accurate motion models at this moment with more input measurements

received. The average error bound at stage **I** is higher than that at stage **III** due to the fact that we do not know when the motion pulse will start and we have to widen the error bound according to the amount of acceleration clipping used.

To study the performance of our method, we have implemented a reference method. This method again divides each motion pulse into three stages. At each stage, it constructs the upper and lower error bounds so that the bound tightness is set to the average value as shown in Table 1. However, unlike our method that changes the bound tightness dynamically, the reference method has a fixed bound tightness throughout the complete stage. As shown in Table 2, the bound tightness at each stage is always the same.

Table 1. The statistical results of our method.

		Upper Deviation	Lower Deviation	Bound Tightness
I	Minimum	5.8%	2.2%	9.9%
	Average	11.3%	6.8%	16.8%
	Maximum	22.5%	11.4%	26.9%
II	Minimum	4.2%	1.6%	5.9%
	Average	12.3%	8.2%	19.8%
	Maximum	30.5%	17.5%	40.0%
III	Minimum	3.3%	2.4%	7.7%
	Average	9.3%	6.1%	14.9%
	Maximum	27.9%	12.2%	34.6%

Table 2. The statistical results of the reference method.

		Upper Deviation	Lower Deviation	Bound Tightness
I	Minimum	4.2%	7.1%	16.8%
	Average	7.7%	10.7%	16.8%
	Maximum	13.6%	16.1%	16.8%
II	Minimum	1.6%	4.3%	19.8%
	Average	7.9%	12.6%	19.8%
	Maximum	17.8%	21.5%	19.8%
III	Minimum	2.2%	2.8%	14.9%
	Average	6.5%	9.2%	14.9%
	Maximum	15.5%	15.1%	14.9%

To compare the performance of the two methods, we have measured the percentages of measurements (actual positions) that fall outside of the error bounds using our method and the reference method as shown in Tables 3 and 4, respectively. Again, all the statistical results are computed separately within each stage of the motion pulses. The definitions of the terms are as follows:

$$\text{Lower Bound Error (\%)} = \frac{\text{no. of measurements exceeding lower bound}}{\text{total no. of measurements}}$$

$$\text{Upper Bound Error (\%)} = \frac{\text{no. of measurements exceeding upper bound}}{\text{total no. of measurements}}$$

Table 3. The reliability of our method.

	I	II	III
Lower Bound Error	6.8%	3.1%	3.9%
Upper Bound Error	8.3%	11.5%	6.9%
Total Error	15.1%	14.6%	10.8%

Table 4. The reliability of the reference method.

	I	II	III
Lower Bound Error	6.8%	5.2%	5.5%
Upper Bound Error	9.9%	9.7%	8.0%
Total Error	16.7%	14.9%	13.5%

From Table 3, we can see that the maximum error of our method occurs at stage I, where the error is 15.1%, i.e., an average of 15.1% of the actual measurements within stage **I** falling outside of the upper or lower bound, when the prediction length is 250ms. In general, our method performs better than the reference method in all three stages, with lower total errors. Note that the reference method requires the knowledge of the total displacement of the motion pulse in order to set the upper and lower bounds, which is not possible in practice, while our method does not require any prior knowledge. We have also conducted the experiment with the prediction length set to 1 prediction step, our results show that the percentage of measurements falling outside of the bounds using our method drops to only 2.3%.

5.5 Experiment 5

In order to show the performance of our method in estimating the error bound in a real application, we have incorporated it into an area-based visibility culling algorithm for virtual walkthrough [17]. With a point-based visibility culling algorithm, a potential visibility set (PVS) is computed for rendering based on a given view point [8]. However, if all occluders visible from a given location (360^0) are shrinked by an ε-distance, the resulting PVS will be valid even if the viewer moves inside a circular region of radius ε-distance centered at the given location [17]. With such an area-based visibility culling algorithm, we no longer need to compute the PVS at every step as the user navigates in a virtual environment. We only need to compute the PVS (with occluders shrinked by ε-distance) at 2ε apart.

Here we may apply our method for estimating the error bound to determine the ε-distance of the area-based visibility culling algorithm. Figure 14 shows a snapshot of distributed virtual environment, which is based on dynamic geometry streaming [7]. Given the current location of the viewer, we predict the viewer's location after 5 prediction steps, i.e., 0.25s. We also estimate the upper and the lower bounds. We then compute D_u, the Euclidean distance between the upper bound and the predicted location, and D_l, the Euclidean distance between the predicted location and the lower bound. Depending on which one is larger, we then select D_u or D_l as ε-distance for computing the PVS. The server will then send to the client all the geometry primitives within the PVS that have not been sent. The idea here is that if the error bound is a good estimation of the prediction error, we should be able to prefetch just enough geometry for rendering, without over-transmission.

As a comparison, we have created a reference method that uses the residual between the latest actual location of the viewer received and the corresponding predicted location to determine ε-distance for the next predicted location. (This is similar to the approach used in the Kalman Filter.) Figure 15 shows the cache hit rates and the bandwidth usages of the two methods during a walkthrough sequence. The two curves at the top show the hit rates of the two methods. We can see that occasionally, the hit rates of both methods are 100%, which means that all the geometry needed for rendering are already available at the client's local cache at the time. This typically happens when the viewer is walking along a straight road, where the prediction accuracy can be very high. However, occasionally, the hit rates drop due to some missing geometry. This typically happens when the viewer is turning at a corner, where some obscured buildings may suddenly become visible and our predictor may fail to prefetch some of the geometry to the local cache in advance.

Judging from the two hit rate curves, it may appear that the two methods have similar performance. In fact, the average hit rates of the two methods are very close (97.76% for the reference method and 97.89% for our method). However, if we compare the bandwidth consumption of the two methods as shown at the

bottom of Figure 15, the situation is very different. The reference method has significantly higher bandwidth consumption than ours. This means that it requests far more buildings from the server than ours. The reason for this is that the reference method tends to use a much higher ε-distance for computing the PVS, due to the high residual values in general. However, such a high bandwidth consumption but with similar hit rate of the reference method suggests that most of the requested buildings are unnecessary and hence the ε-distance used for computing the PVS is larger than necessary.

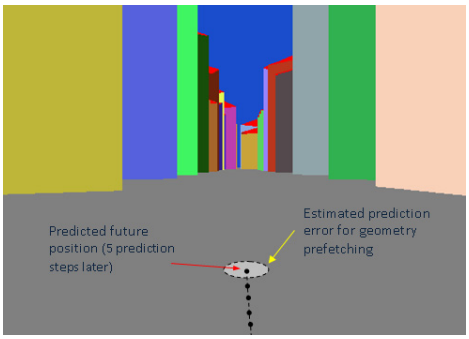

Figure 14. A user view, with predicted position and ε-distance.

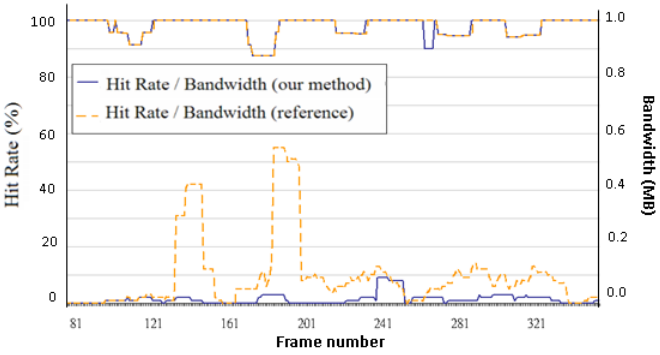

Figure 15. Hit rate (upper curves) and the bandwidth consumption (bottom curves).

6 Conclusion

In this paper, we have discussed the importance of motion prediction in CVEs and the importance of being able to estimate the error bound of each prediction made by the motion predictor. To do this, we have identified four sources of prediction error. They are mismatch of motion model, noise, quantization and time precision. To construct reliable error bounds, we propose some methods to help reduce the prediction error to a "boundable" level. In particular, we propose to clip the motion acceleration for the first few samples, before the client is aware of the start of a new motion pulse. We also propose to reduce the inaccuracy in sampling time by time-stamping each input measurement. By reducing the prediction error with these methods, we divide a motion pulse into three stages and propose methods to estimate the upper and the lower error bounds at each of the three stages.

We have shown the lower and upper error bound curves computed by our method, under low (50ms) and high (250ms) network latency. We can clearly see that at high network latency, the predicted curve fluctuates more and the prediction error increases. As a result, the upper and lower error bound curves need to be widened. We have also compared our method with a reference method, which has prior knowledge of the total displacement of the motion pulse. Results show that our method performs better than the reference method. Using error bounds of the same tightness, our method produces lower error rates than the reference method. The walkthrough experiment further shows that at the same hit rate, our method can produce a tighter error bound, resulting in lower bandwidth consumption due to the reduction on data requests, as compared with the reference method.

Acknowledgements

The work described in this paper was partially supported by a GRF grant from the Research Grants Council of Hong Kong (RGC Reference No.: CityU 116008).

References

[1] J. Ahrens and K. Brislawn, "Large-Scale Data Visualization using Parallel Data Streaming," *IEEE Computer Graphics and Applications*, **21**(4):34-41, July 2001.

[2] R. Azuma and G. Bishop, "A Frequency-Domain Analysis of Head-Motion Prediction," *Proc. ACM SIGGRAPH*, pp. 401-408, 1995.

[3] T. Capin, I. Pandzic, N. Magnenat-Thalmann, and D. Thalmann, "A Dead-Reackoning Algorithm for Virtual Human Figures," *Proc. IEEE VRAIS*, pp. 161-169, 1997.

[4] A. Chan, R. Lau, and B. Ng, "A Hybrid Motion Prediction Method for Caching and Prefetching in Distributed Virtual Environments," *Proc. ACM VRST*, pp. 135-142, Nov. 2001.

[5] A. Chan, R. Lau, and B. Ng, "Motion Prediction for Caching and Prefetching in Mouse-Driven DVE Navigation," *ACM Trans. on Internet Technology*, **5**(1):70-91, Feb. 2005.

[6] A. Chan, R. Lau, and L. Li, "Hand Motion Prediction in Distributed Virtual Environments," *IEEE Trans. on Visualization and Computer Graphics*, **14**(1):146-159, Jan. 2008.

[7] J. Chim, M. Green, R. Lau, H. Leong, and A. Si, "On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments," *Proc. ACM Multimedia*, pp. 171-180, 1998.

[8] D. Cohen-Or, Y. Chrysanthou, C. Silva, and F. Durand, "A Survey of Visibility for Walkthrough Applications," *IEEE Trans. on Visualization and Computer Graphics*, **9**(3):412-431, 2003.

[9] D. Hanawa and T. Yonekura, "Improvement on the Accuracy of the Polynomial Form Extrapolation Model in Distributed Virtual Envrironment," *The Visual Computer*, pp. 369-379, 2007.

[10] J. Huang and H. Tan, "Vehicle Future Trajectory Prediction with a DGPS/INS-based Positioning System," *Proc. American Control Conference*, June 2006.

[11] M. Isard and A. Blake, "CONDENSATION – Conditional Density Propagation for Visual Tracking," *Int'l Journal of Computer Vision*, **29**(1):5-28, 1998.

[12] J. LaViola Jr., "An Experiment Comparing Double Exponential Smoothing and Kalman Filter-Based Predictive Tracking Algorithms," *Proc. IEEE VR*, pp.283-284, 2003.

[13] L. Li, F. Li, and R. Lau, "A Trajectory-Preserving Synchronization Method for Collaborative Visualization," *IEEE Trans. Visualization and Computer Graphics (Proc. IEEE Vis)*, **12**(5):989-996, 2006.

[14] J. Liang, C. Shaw, and M. Green, "On Temporal-Spatial Realism in the Virtual Reality Environment," *Proc. ACM UIST*, pp. 19-25, 1991.

[15] S. Singhal and M. Zyda, *Networked Virtual Environements: Design and Implementation*, ACM Press, 1999.

[16] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *TR 95-041*, Dept. of Computer Science, UNC, 1995.

[17] P. Wonka, M. Wimmer, and F. Silion, "Instant Visibility," *Proc. Eurographics*, **20**(3):411-421, 2001.

[18] J. Wu and M. Ouhyoung, "On Latency Compensation and its Effects on Head-motion Trajectories in Virtual Environments," *The Visual Computer*, **16**(2):79-90, 2000.