A Hybrid Motion Prediction Method for Caching and Prefetching in Distributed Virtual Environments

Addison Chan
addi@cs.cityu.edu.hk

Rynson W. H. Lau
rynson@cs.cityu.edu.hk

Beatrice Ng
beatrice@cs.cityu.edu.hk

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

ABSTRACT

Although there are a few methods proposed for predicting 3D motion, most of these methods are primarily designed for predicting the motion of specific objects, by assuming certain object motion behaviors. We notice that in desktop distributed 3D applications, such as virtual walkthrough and computer games, the 2D mouse is still the most popular device being used as navigation input. Through studying the motion behavior of a mouse during 3D navigation, we propose a hybrid motion model for predicting the mouse motion during a 3D walkthrough. At low motion velocity, we use a linear model for prediction and at high motion velocity, we use an elliptic model for prediction. We describe how this prediction method can be integrated into our distributed virtual environment for object model caching and prefetching. We also demonstrate the effectiveness of the prediction method and the resulting caching and prefetching mechanisms through extensive experiments.

Keywords

Motion prediction, caching, prefetching, 3D navigation, distributed virtual environments, virtual walkthrough.

1. INTRODUCTION

Distributed virtual environments (DVEs) allow geographically separated users to participate or sometimes interact in the same environment over the network. Useful applications of DVEs include remote training, virtual touring, collaborative gaming and collaborative design [3,9,12,15]. However, since DVE systems strongly rely on the network, they suffer from some fundamental problems. The major ones are bandwidth limitation and network latency, in particular when the Internet is used for communications.

There are two main approaches to distribute virtual objects from the server to the clients in DVE applications. Most systems, such as DIVE [6], SIMNET [5], and VLNET [18], employ a complete replication approach to distribute all geometry data to the clients before the start of the application. Since the geometry database is usually large in size, this approach assumes offline downloading or the use of a high-speed network in order to reduce the pre-loading time. Another approach to distribute geometry data is to send them on-demand to the clients at run-time [11,20,21]. The idea of this approach is that when the virtual environment is large, a viewer would likely visit only a small part of it. Hence, it is only necessary to transmit the visible region of the environment to the client to reduce startup time and network traffics. However, to overcome the bandwidth limitation and network latency, a prefetching mechanism is needed to predict objects that will likely be visible to the user shortly and downloads them in advance.

A good prefetching mechanism relies on an accurate motion prediction method, which predicts the future positions of the user. An inaccurate prediction method wastes both network bandwidth and system resources. It may even further reduce the interactivity of the system. In practice, predicting the future, including the user's inputs, is a very difficult and challenging task. It is impossible to take into account of all related factors in the prediction, and any unanticipated events may cause serious prediction errors. Fortunately, it is possible to obtain a reasonably accurate predictor, which can at least improve the overall system performance.

In [7], we presented a general idea of a mouse-based prediction method based on the elliptic model discussed here and some initial results on the accuracy of the prediction method. However, the major limitation of the work is that there are a lot of parameters that need to be fine tuned, making it very difficult for implementation. This paper extends the work significantly in the following ways:

1. We refine the method by introducing a new approach to determine the pulse constants. The new method improves the accuracy of the prediction method and reduces noise interference significantly by capturing the user's recent motion behavior.
2. We show how we may determine the parameters needed for the implementation of the prediction method and how the prediction method can be integrated into our DVE prototype system developed earlier [8,9] to improve the performance of the caching and prefetching mechanisms.
3. We perform extensive experiments on the proposed prediction method with some popular prediction methods and on the performance of the revised caching and prefetching mechanisms, under different testing conditions.

The rest of the paper is organized as follows. Section 2 investigates existing motion prediction methods. Section 3 presents our hybrid motion prediction method. Section 4 discusses techniques for determining the pulse constants and other relevant parameters. Section 5 briefly describes how the prediction method can be integrated into the caching and prefetching algorithms of our DVE prototype system. Section 6 demonstrates the effectiveness of our prediction method and the resulting caching and prefetching mechanisms through various experiments. Finally, section 7 concludes the paper with a discussion on possible future work.

VRST'01, November 15-17, 2001, Banff, Alberta, Canada.

2. RELATED WORK ON MOTION PREDICTION

Although there are many prediction methods proposed, most of them are developed for specific purposes and methods that are developed for navigation prediction are very limited. In this section, we survey different prediction methods that either are developed primarily for or can be adopted to motion prediction.

One kind of path prediction methods is based on the statistical method using random process analysis and past navigation patterns. An example method of this kind is presented in [17] for mobile computing. It considers the fact that most people follow some regular paths for their regular activities, such as going from home to the office. However, a person may occasionally choose to have a path different from his/her regular movement pattern. Two models are proposed in [17] to model these two situations. One is responsible for the recognition of regular paths. The other is responsible for predicting non-regular movement using the Markov model, which is based on Markov process analysis. Unfortunately, the analysis requires collecting a large amount of data and may also involve expensive computations.

Dead reckoning [10] is a popular technique used in DVE systems to reduce bandwidth consumption in transmitting the positional information of moving objects. Each participating machine runs a simulator to extrapolate or predict future states of a moving object from its current state. There are many possible ways to extrapolate the movement of an object. The most popular method is the *polynomial predictor*, which is included in the DIS protocol of IEEE standard 1278. Although the 2nd order polynomial predictor is commonly used, polynomials of other orders may also be used in the predictor:

zero order: $p_{new} = p$

first order: $p_{new} = p + t * v$

second order: $p_{new} = p + t * v + 0.5 * t^2 * a$

where p is the initial position, p_{new} is predicted position, v is the velocity, a is the acceleration, and t is the time at which p_{new} is to be predicted. This method assumes that the motion model is deterministic and follows a simple formula. In [22], a hybrid approach was suggested. The first order polynomial is used if the acceleration is either minimal or substantial; otherwise, the second order polynomial is chosen. Although this method provides a generic model for many kinds of motion, human motion may not behave according to it. In particular, if one moves in a virtual environment and changes the acceleration rapidly, the prediction error could be high.

Another prediction model is the EWMA scheme suggested in our previous work [8,9]. This method predicts future movements based on weighted past movement vectors, with the current vector given a weight of 1 and each preceding vector decreased by a factor of α, $0 \le \alpha \le 1$:

$$\hat{m}_{n+1} = \alpha\ \hat{m}_n + (1-\alpha)\,\vec{m}_n$$

where $\hat{m}_{n+1}$ and $\hat{m}_n$ are the predicted movement vectors for step $n+1$ and n, respectively. $\vec{m}_n$ is the actual movement vector at step n. Although the EWMA scheme can quickly adapt to the change in the user's movement pattern, the predicted movement vectors always depend heavily on the recent movement vectors. Hence, if the user moves in an arbitrary way, the prediction results may become ineffective. In particular, as we have discussed in [7], although the EWMA scheme is reasonably accurate in predicting 3D navigation, it is not too effective in predicting motion of a 2D mouse, which is yet the most popular device used in majority of the systems for navigation input.

The Kalman-based predictor [2,4] is more sophisticated than the *polynomial predictor*. It was originally used to filter measurement noise from linear systems by minimizing the mean square estimation error in a recursive way. In other words, it finds a solution of minimal error, which is the difference between the actual entity state and the observed or measured entity state. It may work with the polynomial predictor or other prediction models to further reduce the prediction errors. Similar to the polynomial predictor or dead reckoning, this method assumes the predicted motion to follow a fixed motion model. Obviously, users' movement patterns can be arbitrary. It is very difficult to find a motion model that fits all situations perfectly. In addition, experimental results in [2] show that during rapid movement, the predictor becomes less effective and its performance is similar to one without using the Kalman filter.

Apart from polynomial-based prediction, there are customized prediction methods for special objects. For example, in [14], the movement of an aircraft is modeled by a circular path because an aircraft typically moves spirally in their designed virtual environment. Another method is the Gauss-Markov process modeling to predict head motion [16]. It is because head motion is usually consisted of some burst changes in viewing angle accompanying with long static viewing direction afterwards. This kind of specialized prediction usually produces more accurate results because it takes into account the motion behavior of the target objects and extracts more information than the polynomial predictor does. The tradeoff is the loss of generality.

To guarantee object availability in distributed walkthrough, [11] maintains at the client a region of the virtual environment around the viewer. This region is larger than the viewing region of the user. As the user moves towards one direction, new objects entering the region may begin to transmit to the client and hence, there is no need to do any prefetching. In [1], a simple method is used to prefetch object models from the secondary memory to the main memory based on viewing direction and velocity of the user. In both methods, the amount of data prefetched is often far higher than necessary. A motion prediction method specifically designed for 3D navigation will certainly help reduce the amount of data needed to be prefetched.

3. A HYBRID MOTION PREDICTION METHOD

In order to accurately predict the mouse motion, we need to construct a reliable motion model for a moving mouse. We adopt the motion model that we developed recently [7]. Figure 1 shows a mouse motion record captured from a sampled navigation step with the vertical component of the mouse motion velocity plotted against time. A typical navigation step contains a number of pulses. A positive pulse represents a forward movement and a negative pulse represents a backward movement. A graph with similar pattern can also be obtained from the horizontal component of the mouse motion velocity.

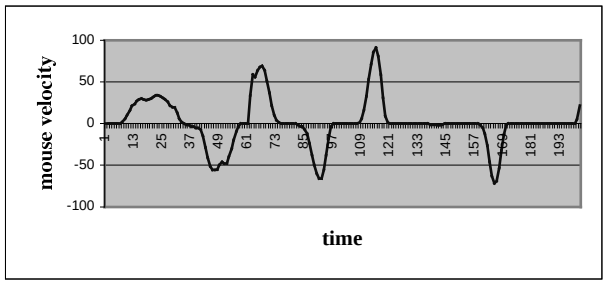


Figure 1. Motion motion velocity obtained from sample navigation.

3.1 An Elliptic Model for High Velocity Motion

Figure 2 shows a plot of the mouse motion acceleration against velocity for one of the positive pulses shown in figure 1. The trajectory of the scattered points can be approximated by an ellipse that starts and ends at the origin, in a clockwise direction. The principle axis of the ellipse lies on the x-axis of the graph and passes through the origin. The ellipse can be formulated as follows:

$$v^2B^2 - 2vAB^2 + A^2a^2 = 0 \quad (1)$$

where v is the velocity, and a is the acceleration (i.e., dv/dt). $2A$ and $2B$ are the lengths of the major and the minor axes of the ellipse, respectively.

For a negative pulse, the ellipse can be formulated as:

$$v^2B^2 + 2vAB^2 + A^2a^2 = 0 \quad (2)$$

We may solve equations (1) and (2) by defining the initial condition v=0 at t=0:

$$v = K_1(\cos(K_2t) - 1) \quad (3)$$

where K_1 and K_2 are arbitrary non-zero constants related to A and B. K_1 is a positive value for a negative pulse and vice versa. The width and height of a pulse determine the absolute values of K_1 and K_2. A wider pulse causes a smaller absolute value of K_2, and a higher pulse causes a higher absolute value of K_1. Since K_1 and K_2 are constants within the period of a pulse and need to be recomputed for every new pulse, we refer to them as the *pulse constants*. Note that t needs to be reset to zero once a new pulse is detected.

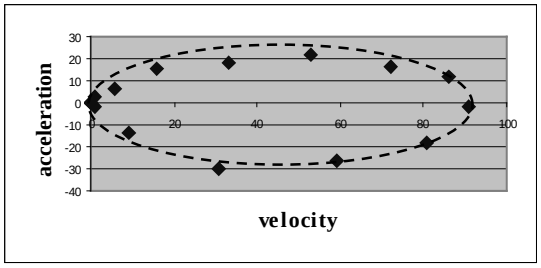


Figure 2. Mouse motion acceleration against velocity for a positive pulse.

3.2 A Linear Model for Low Velocity Motion

We may observe from figures 1 and 2 that most of the sample points are concentrated around the origin if low velocity motion is involved. In this case, it is inappropriate to interpret the trajectory as an ellipse due to the increased significance of noise and distortion in the captured data. So, we model low velocity motion with a generic formula commonly used in dead reckoning:

$$v = v_o + a * \Delta t \quad (4)$$

where v_0 is the initial velocity, v is the velocity after time Δt. When the velocity is high enough, we then switch to equation (3).

4. DETERMINING THE PARAMETERS

4.1 Determining the pulse constants

Due to the possible presence of noise from the measurement, we use the Kalman filter in our predictor to determine the values of the two pulse constants K_1 and K_2 for each pulse. To apply the Kalman filter on equation (3), K_1 and K_2 can be regarded as components of the state vector in the filter. This is different from [2] since our method does not take the velocity and acceleration values as the components of the state vector. As mentioned in Section 3, we use a linear model for prediction at low velocity motion and switch to the elliptic model for prediction once the velocity reaches certain level. However, in order to switch to the elliptic model, we need to determine the values of the pulse constants. Hence, we first determine the preliminary values of the pulse constants and then use them as the state vector of the Kalman filter. Let v_1 and v_2 be the first and the second velocity values, respectively, taken since the start of a pulse. (In fact, zero should be the first velocity value, but it cannot be used to evaluate the preliminary values of the pulse constants.) So, we have:

$$v_1 = K_1(\cos(K_2(1)) - 1) \quad (5)$$

$$v_2 = K_1(\cos(K_2(2)) - 1) \quad (6)$$

By solving these two equations, we obtain

$$K_1 = 2\,v_1^2 / (v_2 - 4\,v_1) \quad (7)$$

$$K_2 = \cos^{-1}(v_2 / (2\,v_1) - 1) \quad (8)$$

If v_1 and v_2 taken are not valid, such as when one of them is zero, the system will continue to use the linear model for prediction and v_1 and v_2 taken previously are ignored. Sometimes, t may reach $2\pi / K_2$. If this happens, the predictor will regard the pulse as terminated and wait for the start of a new pulse.

As mentioned in the previous section, most of the sample points concentrate around the origin if low velocity motion is involved. In this situation, if we interpret the data points with our elliptic model in order to determine the preliminary values of the pulse constants, the increased significance of noise and distortion in the data points may increase the error in estimating the values of the pulse constants. This error will affect the prediction accuracy for the complete pulse period as in our earlier work [7]. However, the error can be reduced if we adjust the values of the pulse constants by the characteristics of the user's past motion behavior. Since the pulse constants determine the height and width of a pulse, to reduce the error in estimating the preliminary values of the pulse constants, we adjust the values of the pulse constants obtained from equations (7) and (8) as follows:

$$K_1^* = K_1 (\delta_1 + \beta_1 \bar{v}) \quad (9)$$

$$K_2^* = K_2 (\delta_2 + \beta_2 / \bar{l}) \quad (10)$$

where K_1^* and K_2^* are the adjusted pulse constants. δ_1, δ_2, β_1 and β_2 are some constants. $\bar{v}$ is the EWMA absolute velocity determined in the last prediction step and $\bar{l}$ is the width of the EWMA pulse obtained from the last motion pulse. $\bar{v}$ and $\bar{l}$ can be calculated in a way similar to our EWMA scheme [8]:

$$\bar{v} = \begin{cases} \alpha_1 \hat{v} + (1-\alpha_1)v & \text{if } v \neq 0 \\ \text{unchange} & \text{otherwise} \end{cases} \quad (11)$$

$$\bar{l} = \alpha_2 \hat{l} + (1-\alpha_2)l \quad (12)$$

where α_1 and α_2 are the weight constants, v is the distance moved by the mouse during the last step (i.e., the last sampling interval). l is the width of the last motion pulse, $\hat{v}$ and $\hat{l}$ are in fact $\bar{v}$ and $\bar{l}$ computed in the last step. After the sampling of the mouse velocity at each step, we compute equation (11) if v is not zero; otherwise, $\bar{v}$ is unchanged. After the end of each pulse, we compute equation (12) to update the value of $\bar{l}$. From equations (9) and (10), K_1^* increases

with increasing $\bar{v}$ while K_2^* decreases with increasing $\bar{l}$, to capture the past motion behavior of the user. In addition, we use the EWMA values of v and l instead of their mean values to adjust the values of the pulse constants because EWMA may adapt quicker to the change in the motion characteristics of the user. We demonstrate the improvement of this approach to [7] in section 6.

4.2 Determining other parameters

Typically, the process noise covariance matrix Q used in a predictive Kalman filter is fixed. Here, we use two process noise covariance matrices, Q_H and Q_L. For the first iteration of each pulse, we use Q_H, which has high absolute component values. Then, we switch to Q_L, which has relatively lower absolute component values, for the rest of the pulse. This design is based on the fact that the preliminary values of the pulse constants derived from equations (7) and (8) may introduce a higher error than those generated by the Kalman filter in its steady state. So, we use Q_H to compensate for the higher uncertainty. On the contrary, the measurement error covariance matrix R is fixed.

If we assign the state vector as $\hat{x}_k = (K_1^*, K_2^*)^T$, the update of the state vector in linear form is:

$$\hat{x}_{k+1}^- = \hat{x}_k + u_k \tag{13}$$

where u_k is a vector updated at the first iteration of each pulse and a zero vector for the rest of the pulse. For the evaluation of u_k at the beginning of each pulse, we need not determine it explicitly. We may just evaluate $(K_1^*, K_2^*)^T$ for each new pulse as described above and assign it to $\hat{x}_{k+1}^-$. With measurement update, the Kalman filter will generate $\hat{x}_{k+1}^-$ as the *a priori* estimate for the next step and $\hat{x}_k$ as the *a posteriori* estimate for the current step. The state transition matrix ϕ_k is fixed to I. On the other hand, the measurement process as shown in equation (3) is in a non-linear form and hence, by applying the extended Kalman filter, the measurement update step can be written as:

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-)) \tag{14}$$

where $h(\hat{x}_k^-)$ is a function equivalent to the R.H.S. of equation (3), z_k is the measured velocity value and K_k is the Kalman gain. The matrix corresponding to the noiseless connection between the state vector and the measurement is denoted as H_k. It is determined by:

$$H_k = ((\partial h(x)/\partial x)|_{x=\hat{x}_k^-})^T \tag{15}$$

which can be rewritten as:

$$H_k = (\cos(K_2 t)\text{-}1, \; -K_1 t \sin(K_2 t)) \tag{16}$$

by substituting equation (3) into equation (15). (Please refer to [4,23].)

To determine the values of parameters Q_H, Q_L, R, β_1, β_2, δ_1, δ_2, α_1 and α_2, we apply the Powell's method [19] here by minimizing the absolute error for one step prediction as the objective function:

$$Q_H \approx \begin{pmatrix} 37.852 & 0.991 \\ 0.991 & 47.852 \end{pmatrix} \qquad Q_L \approx \begin{pmatrix} 30.021 & 0.000 \\ 0.000 & 40.011 \end{pmatrix}$$

$$R \approx (2.000) \quad \beta_1 \approx 0.001 \quad \beta_2 \approx 3.448 \quad \delta_1 \approx 0.037$$

$$\delta_2 \approx 0.165 \quad \alpha_1 \approx 0.650 \quad \alpha_2 \approx 0.800$$

For simplicity, the predictors for the horizontal and the vertical components of the mouse motion share this same set of parameters. Moreover, the threshold for switching from the linear model to the elliptic model Δ_V is set to 2.

5. CACHING AND PREFETCHING WITH MOTION PREDICTION

In our earlier communications [8,9], we proposed the MRM (most required movement) scheme in defining the access score of each object. It is based on the observation that the farther an object is from the viewer, the longer it will take for the viewer to move directly in front. Consequently, the value of caching this object in the local storage or prefetching it from the server is lower. Similarly, the larger the angular distance of an object is from the viewer's line of sight, the value of caching or prefetching the object is also lower. Figure 3 shows this relationship, with D_o representing the distance of object o from viewer v and θ_o representing the angular distance of o from v's line of sight.

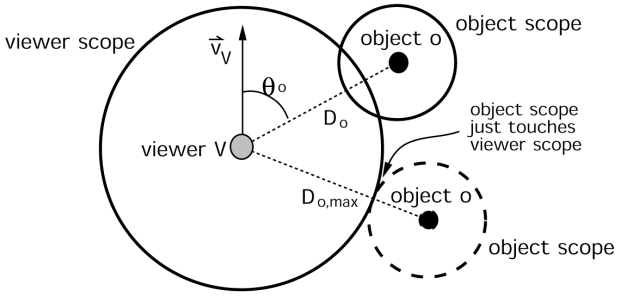


Figure 3. Determining the access score of an object.

To determine the access score $S_{o,v}$ of object o to viewer v:

$$S_{o,v} = \alpha(1 - \frac{D_o}{D_{o,\max}}) + (1-\alpha)(1 - \frac{|\theta_o|}{\pi}) \tag{17}$$

where D_o is the distance between v and o, $D_{o,\max}$ is the sum of radii of the viewer scope and of the object scope, and α $(0 \le \alpha \le 1)$ is a parameter to determine the contributions of D_o and θ_o to $S_{o,v}$.

In our DVE prototype, we employ a similar navigation mapping as a typical VRML browser. We map the vertical mouse motion to the user's forward or backward velocity and horizontal mouse motion to the user's rotational velocity, i.e., change of line of sight, in the 3D environment.

In our earlier work, we use the EWMA scheme to predict where the viewer will be in the next step. The client will then request the server for objects that the viewer is expected to see if it does indeed move to the predicted location. This prefetching operation will increase the availability of objects at the client. However, due to the dramatic increase in error as we attempt to predict more steps ahead using the EWMA scheme, we could only prefetch objects one step ahead. While this still increases the prefetching accuracy, the objects requested may not have reached the client in time for rendering the next frame as the network latency increases or the network bandwidth reduces. This limits the usefulness of the prefetching algorithm. In

addition, due to the limitation in accuracy of the EWMA scheme, we did not use the predicted location for object caching. Instead, we simply cached objects based on the current location of the viewer. This caching strategy has the problem that when the cache is full, the caching algorithm will remove objects based on the current location of the viewer. However, it is possible that objects considered as visually less important at the current location and removed from the cache may become visually important in the next steps.

As shown in the next section, the proposed prediction method is very accurate even when we predict a few steps ahead. Hence, we have revised both the caching and the prefetching algorithms of our DVE prototype to use the proposed prediction method. We have experimented the accuracy of the caching and prefetching algorithms with different numbers of prediction steps. As will be shown later, increasing the number of prediction steps can improve the cache hit ratio significantly.

6. RESULTS AND DISCUSSIONS

We have implemented 3 predictors, P_1, P_2 and P_3, for testing. All of them are operated with a Kalman filter. P_1 denotes our prediction method proposed here. P_2 and P_3 are two popular predictors. P_2 is a predictor using the Guass-Markov process modeling and P_3 is a second order polynomial predictor. We have integrated all these three predictors into the caching and prefetching algorithms in our DVE prototype [9] for comparison. During all our experiments, the client module, which contains the three predictors, ran on a PC with a Pentium III 800MHz CPU. The server module ran on a Sun workstation with an UltraSPARC-IIi 270MHz CPU.

Figure 4 compares the computational costs of the 3 predictors. We observe that P_3 has the highest computational cost. This is because the state vector and the measurement vector have a size of 3 and 2, respectively. It is greater than those in P_2, which have the size of 2 and 2. However, P_1 has the smallest state vector and measurement vector sizes of 2 and 1, respectively. Since the computational cost increases significantly as the sizes of the two vectors in the Kalman filter increase, our method is the fastest of all three methods. Another reason is that in our method, we only need to run the expensive Kalman filter when the input absolute velocity value is higher than the threshold value Δ_V. In a typical walkthrough, the user often tends to keep the mouse steady or move it very slightly most of the time. In this situation, the input absolute velocity value will be smaller than Δ_V and hence, P_1 will have a much lower computational cost.

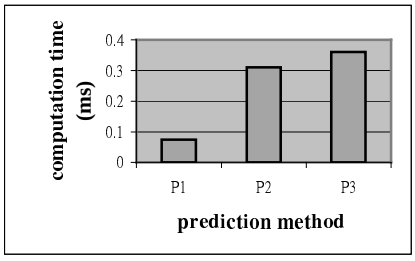


Figure 4. Comparison of computational costs.

We have conducted some navigation experiments using our DVE prototype with the three prediction methods. Four types of navigation patterns are experimented as shown in Figure 5. Figure 5(a) shows the *CP* pattern, which models a constant circular translation pattern. Figure 5(b) shows the *CCP* pattern, which models the same pattern as CP except that the moving direction changes less rapidly at each step. Figure 5(c) shows the *RW* pattern, which models a random walk around the environment. Here, we further divide the *RW* pattern into two types. One has a fast navigation speed, called *RW1*, and the other has a slow navigation speed, called *RW2*. The mouse motion is predicted separately by the three predictors P_1, P_2 and P_3 in the experiments. The predicted mouse velocity is then mapped to the translation and angular velocities in the 3D virtual environment. Our earlier EWMA prediction method, which is not suitable for predicting mouse motion, is also investigated in some of the experiments.

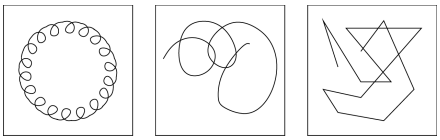


Figure 5. Navigation patterns: (a) CP, (b) CCP, and (c) RW.

In our experiments, we compare the effect of determining the access scores based on current locations and on the predicted locations. We have also investigated the performance of the system with and without prefetching, through measuring the prediction error and hit rate.

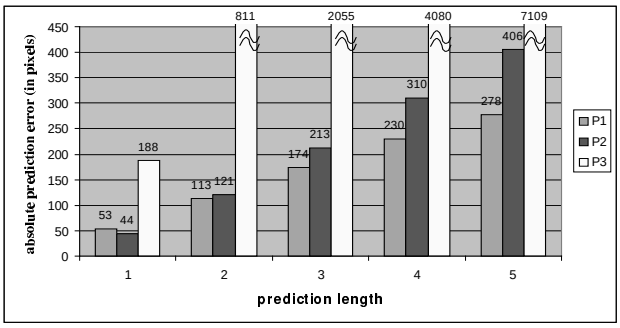
6.1 Prediction error

In order to have a closer investigation on the improvement achieved by equations (9), (10), (11) and (12), we compare our earlier method [7] with P_1. The result is shown in table 1. The significant improvement of the new method over our original method demonstrates the importance of the initial guess of the pulse constants on the accuracy of the prediction.

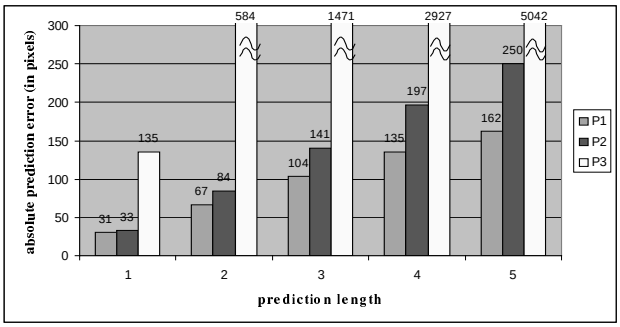
Table 1. Comparison of prediction errors (in pixels).

	Original Method	New Method
CP	80	53
CCP	53	31
RW1	263	121
RW2	18	14

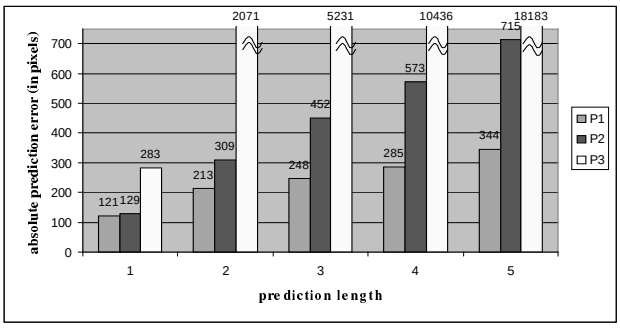
We have also investigated the accuracy of the three predictors, P_1, P_2 and P_3, at different number of prediction steps, i.e., predicting different numbers of steps ahead. We experimented 1 to 5 prediction steps with each of the four navigation patterns: CP, CCP, RW1 and RW2. Note that the parameters used in P_2 and P_3 are also determined by the Powell's method, in a way similar to those determined for P_1. Each prediction step is set to 0.1s and the navigation system reads the predicted location every 0.1s. The error values are the average absolute difference between the actual and the predicted mouse displacement in the vertical direction. (The error values for the horizontal direction are expected to be similar and can be determined in a similar way.) Another possible error measurement may be the absolute distance error. However, using the velocity difference as the error measurement makes the analysis of the predictors easier since the prediction is based on the velocity vectors. In addition, the results from both x and y directions are similar.



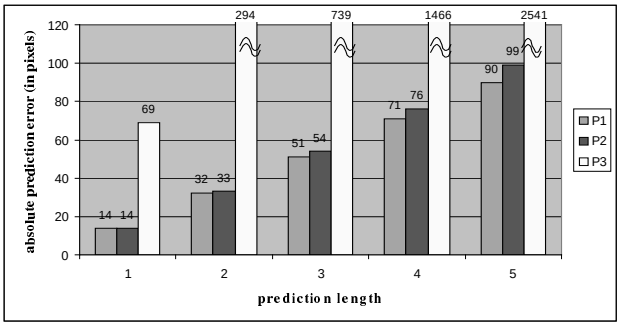
(a) CP



(b) CCP



(c) RW1



(d) RW2

Figure 6. Comparison of prediction errors with different navigation patterns.

From figure 6, we observe that P_3 may be the most inaccurate method among the three. It is pointed out by [2] that the accuracy of a polynomial-based predictor is highly dependent on the prediction length l and the contribution of high frequencies in its signal spectrum. The poor performance of P_3 shows that l or the high frequencies contribution may be too high for this method. It is believed that the performance may improve significantly by reducing the sample size. This can be regarded as a reduction on l or the high frequencies contribution since the velocity value should be lowered if the time unit is decreased. Then, the general variation of the values, and hence the high frequencies contribution, is reduced. From another point of view, l is reduced because of the shortening of the prediction step size.

However, it is difficult to have a high sampling rate of the mouse location, especially under Java implementation. In addition, increasing the sampling rate will lead to a greater fluctuation in the actual sampling time. Figure 7(a) shows a sketch of the equation of the second order polynomial predictor, where p=0, v=5 and a=15. As the velocity value increases, the slope of the curve becomes steeper and steeper. This does not match with the shape of the pulses shown in Figure 1. Therefore, the generic polynomial predictor is not suitable for predicting mouse motion, compared with P_1 and P_2 .By comparing P_1 in figure 7(c) and P_2 in figure 7(b), we observe that they are similar in accuracy when the prediction step is low. However, as we increase the prediction step, P_1 becomes more and more outstanding than P_2. This can be easily explained by comparing the outputs of the equations. According to the transition matrix in P_2, we know that the equation for evaluating v is:

$$v_{new} = v + a * (\frac{1}{\beta}(1 - e^{-\beta t})) \qquad (18)$$

where β is some constant. The curve is plotted in Figure 7(b), with v=5 and a=15. It is similar to the starting of a pulse shown in Figure 2. However, as we increase the prediction step, the slope becomes flatter and flatter, which causes the predicted value to move towards a constant value. On the other hand, our proposed predictor drops to zero eventually. This matches very nicely with the pulse shown in Figure 2. This accounts for the outstanding performance of P_1.

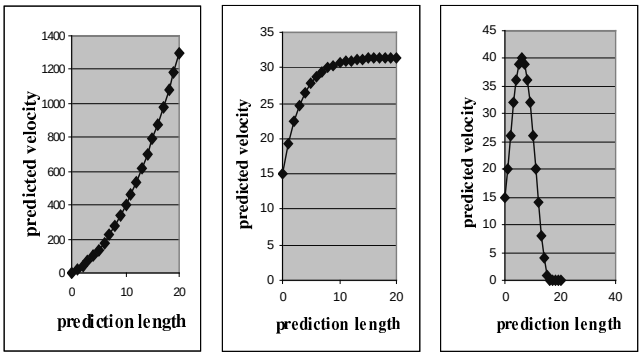


Figure 7. A sketch of: (a) second order polynomial predictor, (b) Gauss-Markov model predictor, and (c) our proposed predictor.

6.2 Hit ratio

To study the performance of the three predictors under a real situation, we have performed some experiments on the prototype system [9]. The system employs a multi-resolution modeling technique similar to the progressive meshes [13] for model transmission. The access scores of all potentially visible objects in the environment are computed at each step to determine the objects for caching, the objects for prefetching and the resolution of each object for rendering, so as to increase the availability of objects and minimize the transmission and rendering costs. The virtual environment that we use for our experiments here has a size of 10,000x10,000 square units,

containing a total of 5,000 objects. The average size of each object is about 300KB. The total database size is approximately 1.5GB. The cache size is fixed at 4MB.

The performance of the three motion predictors can be measured by the cache hit ratio. Cache hit ratio refers to the percentage of bytes of objects inside the viewing region of the viewer available from the client's local cache at the time of rendering. A high cache hit ratio implicitly indicates that the output images have high visual quality.

We have performed three experiments with the prototype system. In the first experiment, we tested the performance of the caching and prefetching mechanisms with different prediction methods. In the second experiment, we studied the behavior of the prefetching mechanism by varying the number of prediction steps. In the last experiment, we tested the performance of the caching and prefetching mechanisms with different navigation patterns.

6.2.1 Experiment #1

In the first experiment, we investigate the performance of the caching and prefetching mechanisms with different prediction methods. The prediction methods that we have experimented include P_1, P_2, P_3 and the EWMA method. The weight of the EWMA method is set to 0.5. Figure 8 shows the results of the experiment. Result #1 is a base case for comparison - no prefetching is preformed and the access score is determined based on the current location. In results #2 to #5, we use the four prediction methods to predict the viewing location and direction in the next step, which are then used to determine the access scores of objects for caching. However, no prefetching is performed. Results #6 to #9 are collected under the same conditions as results #2 to #5, except that prefetching is performed this time based on the predicted viewing location and direction.

Result	Prefetching	Access Score Computed at
# 1	No	Current Viewing Location
# 2	No	EWMA Predicted Location
# 3	No	P1 Predicted Location
# 4	No	P2 Predicted Location
# 5	No	P3 Predicted Location
# 6	Yes	EWMA Predicted Location
# 7	Yes	P1 Predicted Location
# 8	Yes	P2 Predicted Location
# 9	Yes	P3 Predicted Location

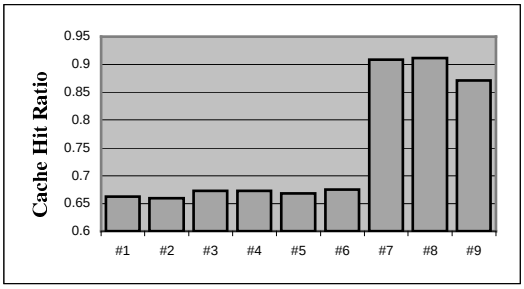


Figure 8. Results of Experiment #1

From figure 8, we may observe that even if there is no prefetching performed in result #1, the hit ratio is reasonably high, due to the small incremental change in object visibility between consecutive frames and the satisfactory performance of the caching mechanism. Results #3 to #5 generally show slightly higher hit ratios than result #1. This is because when the prediction methods are accurate enough, using the predicted location to compute the access score helps determine future object visibility more accurately. Objects currently in the cache but with low predicted visibility will be reduced in size or removed from the cache by the caching mechanism. This approach, however, is risky because if the prediction method is not reliable, objects that are actually needed in the immediate future may be removed. Result #2 shows that the EWMA scheme may not be reliable enough for this purpose as it actually produces a lower hit ratio than result #1. In general, results #3 to #5 are very close because all of them do not perform prefetching, but we can see that P_1 and P_2 produced a slightly higher hit ratios as they are more accurate in predicting mouse position than P_3.

Results #6 to #9 generally achieve much higher hit ratios than results #2 to #5. However, we note again that the performance of the EWMA scheme is not too good. There is only a small improvement on hit ratio when prefetching is performed. In contrast, results of P_1, P_2 and P_3 show bigger improvements on hit ratio after perfetching is performed. However, P_1 and P_2 have higher hit ratios then P_3, indicating that they are more accurate in predicting object visibility in the next step.

6.2.2 Experiment #2

In the second experiment, we investigate the performance of the predictors in predicting a few more steps ahead and the performance of the resulting prefetching mechanism. We test the three predictors P_1, P_2 and P_3 in predicting from 1 to 5 steps ahead. That is, the system will try to prefetch objects which may be accessed in the subsequent 5 steps whenever the network is idle. Objects that are predicted to be visible very shortly will be transmitted first. In figure 9, we may see that the hit ratios of all three methods increase with increasing prediction and prefetching steps. It is because if the system can request for objects earlier in time, more renderable objects will be available immediately from the local cache at the time of the rendering. Moreover, the accuracy of P_1 and P_2 helps to generate a higher hit ratio in all steps then P_3. P_1 does particularly well when the prediction steps are high due to its comparatively high prediction accuracy at high prediction steps.

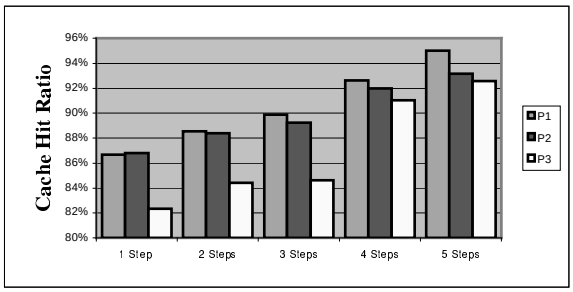


Figure 9. Results of Experiment #2

6.2.3 Experiment #3

In the third experiment, the performance of the three predictors under different moving patterns is evaluated. We have experimented the same four navigation patterns: CP, CCP, RW1 and RW2. Figure 10 shows the resulting hit ratios under the four moving patterns when all the predictors attempt to predict the viewer's location and direction 5

steps ahead. We may observe that the hit ratios obtained from the RW1 pattern are generally lower than those from other navigation patterns. This is because under the RW1 pattern, the viewer moves more rapidly in a random manner, reducing the accuracy of the prediction. On the other hand, our method does better than the other two predictors under all navigation patterns. This is due to its higher accuracy in predicting the mouse motion velocity.

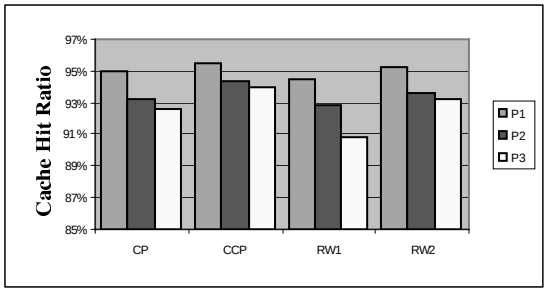


Figure 10. Results of Experiment #3

7. CONCLUSIONS

In this paper, we have described our hybrid method for predicting navigation movement using a mouse-based input device. We have introduced the method and described how to determine all the relevant parameters. We have also discussed how we may integrate the prediction method into the DVE prototype system that we have developed. Finally, we demonstrate the effectiveness of the proposed method as compared with some of the well-known motion prediction methods. Our method does very well under single and multiple prediction steps. Our experiments also demonstrate that the proposed method improves the performance of the caching and prefetching mechanisms significantly under some typical navigation patterns. We are now investigating how our proposed prediction method can be applied to other input devices, in particular the CyberGloves.

ACKNOWLEDGMENTS

The work described in this paper was partially supported by a CERG grant from the Research Grants Council of Hong Kong (RGC Reference Number: CityU 1080/00E).

REFERENCES

[1] D. Aliaga, et al.. "MMR: An Interactive Massive Model Rendering System Using Geometric And Image-Based Acceleration." In *Proc. of ACM Symp. on Interactive 3D Graphics*, pages 199-237, 1999.

[2] R. Azuma and G. Bishop. "A Frequency-Domain Analysis of Head-Motion Prediction." In *Proc. of ACM SIGGRAGH'95*, pages 401-408, 1995.

[3] C. Babski, et al.. "The COVEN Project: Exploring Applicative, Technical, and Usage Dimensions of Collaborative Virtual Environments." *Presence*, **8**(2):218-236, 1999.

[4] R. Brown and P. Hwang. *Introduction to Random Signals and Applied Kalman Filtering* (3rd Ed.). John Wiley & Sons, 1997.

[5] J. Calvin, A. Dicken, B. Gaines, P.Metzger, D. Miller, and D. Owen. "The SIMNET Virtual World Architecture." In *Proc. of IEEE VRAIS*, pages 450-455, 1993.

[6] C. Carlsson and O. Hagsand. "DIVE-a Multi-User Virtual Reality System." In *Proc. of IEEE VRAIS*, pages 394-400, 1993.

[7] A. Chan, R.W.H. Lau, and A. Si. "A Motion Prediction Method for Mouse-Based Navigation." In *Proc. of CGI'2001*, IEEE Computer Society Press, pages 139-146, July 2001.

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

[9] J. Chim, R.W.H. Lau, A. Si, H.V. Leong, D. To, M. Green, and M. Lam. "Multi-Resolution Model Transmission in Distributed Virtual Environments." In *Proc. of ACM VRST*, pages 25-34, 1998.

[10] DIS Steering Committee. "IEEE Standard For Distributed Interactive Simulation - Application Protocols." *IEEE Standard 1278*, 1998.

[11] J. Falby, M. Zyda, D. Pratt, and R. Mackey. "NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation." *Computers & Graphics*, **17**(1):65-69, 1993.

[12] C. Greenhalgh and S. Benford. "Supporting Rich And Dynmic Communication In Large-Scale Collaborative Virtual Environments." *Presence*, **8**(1):14-35, 1999.

[13] H. Hoppe, "Progressive Meshes", In *Proc. of ACM SIGGRAPH'96*, pages 99-108, 1996.

[14] A. Katz and K. Graham. "Dead Reckoning for Airplanes in Coordinated Flight." In *Proc. of Workshop on Standards for Interoperability of Defense Simulations*, pages 5-13, Mar. 1994.

[15] F. Li, R.W.H. Lau, and F. Ng. "Collaborative Distributed Virtual Sculpting." In *Proc. of IEEE VR*, pages 217-224, Mar. 2001.

[16] J. Liang, C. Shaw, and M. Green. "On Temporal-Spatial Realism in the Virtual Reality Environment." In *Proc. of ACM UIST'91*, pages 19-25, 1991.

[17] G. Liu and G. Maguire Jr. "A Class of Mobile Motion Prediction Algorithms for Wireless Mobilecomputing and Communications." *Mobile Networks and Applications*, **1**(2):113-121, 1996.

[18] I. Pandzic, T. Capin, E. Lee, N. Thalmann, and D. Thalmann. "Flexible Architecture for Virtual Humans in Networked Collaborative Virtual Environments." In *Proc. of Eurographics'97*, **16**(3):177-188, 1997.

[19] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. Numerical Recipes in C (2nd Ed). Cambridge University Press, 1992.

[20] D. Schmalstieg and M. Gerautz. "Demand-Driven Geometry Transmission for Distributed Virtual Environments." In *Proc. of Eurographics'96*, **15**(3):421-432, 1996.

[21] G. Singh, L. Serra, W. Png, and H. Ng. "BrickNet. A Software Toolkit For Network Based Virtual World." *Presence*, **3**(1):19-34, 1994.

[22] S. Singal and D. Cheriton. "Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality." *Presence*, **4**(2):169-193, 1995.

[23] G. Welch and G. Bishop. "An Introduction to the Kalman Filter" Available from http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html