# A Motion Prediction Method for Mouse-Based Navigation

Addison Chan [1] Rynson W.H. Lau [1] Antonio Si [2]

[1] *Department of Computer Science, City University of Hong Kong, Hong Kong*
[2] *Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, U.S.A.*

## Abstract

*A distributed virtual reality system allows remote users to share and to view a common virtual environment via connected networks. However, network latency and bandwidth are often the most crucial performance bottlenecks. We have recently developed a distributed virtual walkthrough environment that supports on-demand model transmission over the Internet through the use of, in addition to other techniques, a simple prefetching technique called EWMA. Although the prefetching technique has been shown to be effective in predicting 3D motion during our simulation experiments, it is less effective in our prototype experiments. The main reason is that most input devices used for navigation are 2D in nature (mostly 2D mice) and EWMA is not too effective in predicting user motion in moving a 2D mouse. To overcome this limitation, we propose in this paper a method for predicting the user motion in moving the mouse during a 3D navigation. To improve the accuracy of the prediction, we also propose a constrained navigation method. We will demonstrate the effectiveness of the new method through experimental results.*

## 1. Introduction

Tremendous improvement on 3D graphics has been made over these three decades. Fueled by the popularity of networking technology, computer graphics may now be complemented with computer networks to form a new type of graphics application, called *distributed virtual environments (DVEs)*. Instead of interacting in a virtual environment on a single machine, DVEs allow multiple users at various geographical locations to participate in the same environment over a network, such as the Internet. Useful applications of DVEs include remote training, education, virtual touring, collaborative gaming and collaborative design [3, 8, 11].

Because 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. These problems are mainly due to the typically large in size of the geometry database.

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 [16], 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 the use of a high-speed network in order to reduce pre-loading time. Another approach to distribute geometry data is to send them on demand to the clients [10, 17, 18] at run-time. When the virtual environment is large, a viewer would likely only visit a small section of it. This approach requires only the visible region of the environment to be transmitted to the client and can thus reduce startup time and network traffics. However, due to bandwidth limitation and network latency, a prefetching technique is needed to predict objects that will likely be visible to the user shortly and download them from the server to the client in advance.

A good prefetching mechanism relies on an accurate prediction method, which predicts the future positions of the user. An inaccurate prediction method wastes both network bandwidth and system resources. It even further reduces 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 the factors in the prediction. Any unanticipated events may cause serious prediction errors. Fortunately, it is possible to obtain a reasonably accurate predictor, which can improve the overall system performance.

We have recently developed a distributed virtual walkthrough environment [7, 8] which supports on-demand models transmission over the Internet through the use of, in addition to other techniques, a simple prefetching technique called *EWMA*. Although the prefetching technique has been shown to be effective in predicting user motion in 3D navigation during our simulation experiments, it is less effective in our prototype experiments. The main reason is that most input devices used for navigation are 2D in nature (mostly 2D mice) and EWMA is not too effective in predicting user motion in moving a 2D mouse. To overcome this limitation, we propose in this paper a method for predicting the user motion in moving the mouse during a 3D navigation. We will show towards the end of the paper that the proposed prediction method for 3D navigation is significantly more accurate than existing methods.

The rest of the paper is organized as follows. Section 2 investigates existing prediction methods for navigation. Section 3 presents our motion prediction method together with a constrained navigation algorithm, which improves the accuracy of the prediction. Section 4 briefly describes the implementation of our prototype system. Section 5 compares our prediction method with other methods experimentally. Finally, Section 6 concludes the paper with a discussion on possible future work.

## 2. Related Work

In this section, several popular prediction techniques, which may be employed for navigation prediction, are investigated. Although most of them are not tailor-made for DVE navigation, some of the ideas can still be adopted. For example, systems such as [1, 10] simply prefetch all possible future scene objects according to the viewer's velocity and view angle. The amount of data prefetched is thus far higher than necessary. A motion prediction method specifically designed for 3D navigation is therefore needed to reduce the amount of data prefetched.

One kind of the path prediction method is the statistical method using random process analysis and past walking patterns. It is noted that most users follow some regular paths when going, for example, from home to the office. However, a user may sometimes choose to have a path different from his regular walking pattern. In [14], these two situations are modelled separately. One is responsible for the recognition of regular paths. However, if a user does not follow his/her regular paths, the predictor will estimate the future movements using the Markov model, in which the prediction is based on Markov process analysis. Unfortunately, the analysis requires collecting a large amount of data and may also involve expensive computations.

Dead reckoning [9] is a popular technique used in DVE systems to reduce the network bandwidth in transmitting positional information of moving objects. Each participating machine would run 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. Designers may choose an appropriate polynomial order to use in the predictor, although the 2nd order polynomial predictor is commonly used. The following are some examples:

*zero order:* $p_{new} = p$

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

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

where $p$ is the initial position, $p_{new}$ is the predicted position, $v$ is the velocity, $a$ is the acceleration, and $n$ is time step at which $p_{new}$ is to be predicted. This method assumes that the motion model is deterministic and follows a simple formula. In [19], a hybrid approach was suggested. The first order polynomial is used if the acceleration is either minimal or substantial. Otherwise, the second order one is chosen. Although it is a generic model for many kinds of motion, human motion may not behave according to some generic motion models. In addition, 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 as suggested in our previous work [7, 8]. The model assigns different weights to former movement vectors, with the current vector given a weight of 1 and each subsequent vector decreased by a factor of $\alpha$, $0 \leq \alpha \leq 1$. Hence, the relationship is:

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

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 $n$. The EWMA scheme can avoid using too much memory to store the past user movement history and quickly adapt to the change of the user's moving pattern. However, the predicted movement vectors always depend heavily on the recent movement vectors. If the walking pattern of a user is arbitrary, the prediction results may become ineffective.

The Kalman-based predictor [2, 4] is more sophisticated than the *polynomial predictor*. It was originally used to filter noise from the measurement in a linear system. This is achieved 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 error. Similar to the polynomial predictor or dead reckoning, this method assumes the predicted motion to follow a fixed motion model. Obviously, users' navigation patterns are usually arbitrary. It is very difficult to find a motion model which fits all situations perfectly. In addition, experimental results in [2] show that the predictor performs not so well during rapid movement, and the performance is similar to the one without using the Kalman filter.

Apart from polynomial-based prediction, there are customized prediction methods for specific object motion. For example, in [13], an aircraft's movement is modeled as a circular path because it typically moves spirally in their designed virtual environment. This kind of specialized prediction usually offers more accurate result because it takes into account of the properties of particular objects and extracts more information than the polynomial predictor does. The tradeoff is the loss of generality.

## 3. Motion Prediction with Navigation Constraints

Although some prediction techniques mentioned above are developed for navigation and may have shown to be effective when 3D trackers are used for motion tracking, they are less effective when applied in a desktop environment where 2D mice are often used for motion tracking. This is because when a user uses a 2D mouse as an input device for 3D navigation, a mapping step is needed to map the 2D mouse motion to the corresponding 3D movement. Such a mapping changes the dimension of motion coherence that existing motion prediction techniques rely on in predicting future movement. To overcome this limitation, we propose in this paper a motion prediction method for mouse-based navigation systems. To further improve the accuracy of the prediction, we also propose a constrained navigation algorithm to work with the motion prediction method.

### 3.1. Prediction of Mouse Velocity

In order to accurately predict the mouse motion, we need to construct a reliable motion model for a moving mouse. Figure 1 shows a mouse motion record captured from a sampled navigation step with vertical mouse velocity plotted against time. During the navigation, a number of pulses are generated. 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 (left and right) component.

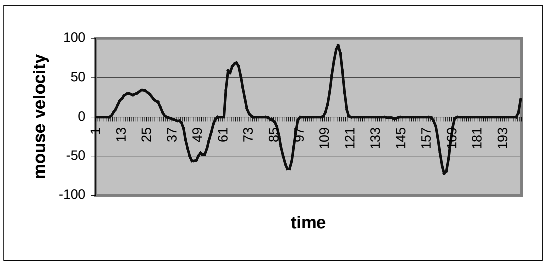

**Figure 1. Mouse motion in a sampled navigation.**

Figure 2 shows a plot of mouse acceleration again velocity for one of the positive pulses shown in figure 1. The trajectory of the scattered points can be approximated by an ellipse which 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:

$v^2\ B^2 - 2\,v^2\ A\ B^2 + A^2\ a^2 = 0$ ……………….(1)

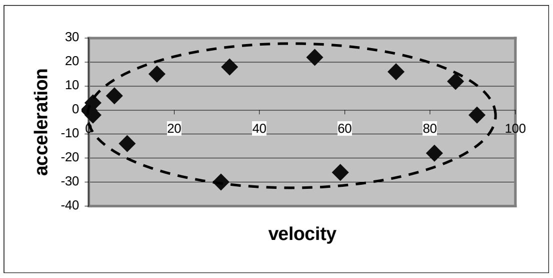

**Figure 2. Acceleration against velocity for a positive pulse.**

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

$v^2\ B^2 + 2\,v^2\ A\ B^2 + A^2\ a^2 = 0$ ……………….(2)

where $v$ is the velocity, and $a$ is the acceleration (i.e., d$v$/d$t$). $2A$ and $2B$ are the lengths of the major and the minor axes of the ellipse, respectively. We may solve equations (1) and (2) by defining the initial condition $v$=0 at $t$=0:

$v = K_1 (\cos(K_2 t) - 1)$ …………..…..………..(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 positive pulse and a negative value for a negative pulse. 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 $K_1$.

From figures 1 and 2, we may observe that most of the sample points will be 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 the noise or the distortion of the motion model. So, we model low velocity motion with a generic formula commonly used in dead reckoning:

$v = v_0 + a^* \Delta t$ ……………………………….(4)

where $v_0$ is the initial velocity, $v$ is the velocity after time $\Delta t$. When the velocity is high enough, we switch to the ellipse model shown in equation (3).

Note that $K_1$ and $K_2$ in equation (3) are recomputed once a new pulse is detected. In addition, *t* also needs to be reset to zero. When the mouse velocity rises above a threshold value, it signals the beginning of a new pulse. On the other hand, when the mouse velocity drops near to zero, it signals the end of the pulse.

## 3.2. Navigation Constraints

A virtual walkthrough is typically made up of arbitrary velocity and acceleration changes, which are hard to predict. The idea of having navigation constraints is that it is not necessary to allow the users to have all the possible freedoms of motion. "Just giving them enough" can still achieve a proper walkthrough without sacrificing the user's perception of smoothness.

Some methods have been proposed to constraint user motion in virtual environments [12, 15]. For example, in a manipulation application [15], the movement speed can be constrained. The speed can be increased when the user is far away from selected object. However, as the user moves near to the object, the moving speed should be reduced. Another example is navigation over a terrain model [12]. The system may constraint the camera orientation to be in line with the slope of the terrain as the user walks on it.

Similar technique can be applied to virtual walkthrough. Instead of having arbitrary velocity and acceleration, we may constraint users' motion velocity and acceleration to certain values, as long as they are not aware of such a constraint. If we can reduce the number of possible motion states, we may perform motion prediction with a greater chance of success.

Our approach to constraining user motion in virtual walkthrough is by reducing the freedom of motion velocity. Let us define the *mouse dragging distance* as the distance of the mouse current position from its position when the user first presses the mouse button to initiate a navigation step. During a virtual walkthrough, the mouse dragging distance *d* is computed and then quantized into an integer number *q*. *q* is then constrained before sending to a mapping process which converts the constrained *q* into the actual translation or angular velocity *c* in the 3D virtual environment. Note that in a typical VRML browser [20], for example, the vertical dragging distance indicates the translation velocity, while the horizontal dragging distance indicated the angular velocity. Both of them are measured in *pixels per unit time*. Since the vertical and horizontal dragging distances are independent of each other, we consider each of them independently and do the prediction separately. For simplicity, we focus our discussion on the prediction of the translation velocity here, without any lost of generality.

The following code segment shows the constrained algorithm:

```
$q_{n-1}$ = Quantization( $d_{n-1}$ );
$q_n$ = Quantization( $d_n$ );
$\Delta q$ = $q_n$ - $q_{n-1}$ ;
if $\Delta q \geq Max$ then
    $c$ = ConstrainedMapping( $q_{n-1}$ +$Max$)
else if $Max > \Delta q \geq 1$ then
    $c$ = ConstrainedMapping( $q_{n-1}$ + 1)
else if $\Delta q = 0$ then
    $c$ = ConstrainedMapping( $q_{n-1}$ )
else if $-1 \geq \Delta q > -Max$ then
    $c$ = ConstrainedMapping( $q_{n-1}$ -1)
else if $-Max \geq \Delta q$ then
    $c$ = ConstrainedMapping( $q_{n-1}$ -$Max$);
```

where *Max* is a customized threshold value, which is greater than 1. ConstrainedMapping() is a monotonic increasing function passing through the origin. With this algorithm, even though we constrain the quantized dragging distance *q*, we may still accelerate to any velocity in at most 2 time units.

This constrained algorithm restricts the number of discrete values that can be reached from the current value *q* to 5 (i.e., *q+Max*, *q+1*, *q*, *q-1*, *q-Max*), except when the mouse is released. In fact, one can adjust the number of possible discrete values. If an application requires more control and freedom, we may increase the number. On the other hand, if an application requires higher prediction accuracy, we may decrease the number. However, there is an exceptional case. If the user stops suddenly by releasing the mouse button, the user's velocity value is set to zero immediately whatever *q* is. It is because stopping suddenly may be a crucial action in navigation, such as preventing from colliding with an obstacle or located the target suddenly. In addition, if the user stops suddenly, no data are required to be transmitted from the server. So, any incorrect prefetching would not increase the latency.

There is an advantage of the constrained navigation. The prediction error is now being bounded to |ConstrainedMapping( $D_{n-1}$ +*Max*) - ConstrainedMapping( $D_{n-1}$ -*Max*)|, ignoring the case of releasing the mouse button. No error is greater than this.

## 4. Implementation

In this section, we look at how we may apply the ideas presented in Section 3 in our motion prediction method. Let us assume that we keep the frame rate of the navigation system at 10 frames per second. The system only needs to sample the mouse location once every 0.1 second. However, in order to have a more accurate prediction of the mouse location at the next time frame, we sample the mouse location once every 0.025 second and we define 0.025 second as *one time unit*. Hence, for each prediction, we need to compute the mouse velocity of the next four time units. They are added up together to become the final predicted mouse position at the next frame. This predicted mouse position is passed to the navigation algorithm to determine the user's future moving velocity in the 3D virtual environment. Note that we have to perform two predictions (horizontal and vertical speed) for each operation. So, both of them have their own $K_1$ and $K_2$. The prediction results are then combined into a final velocity vector.

The Kalman filter is used with the predictor in order to estimate parameters $K_1$ and $K_2$ in the presence of noise, which may come from the elliptic approximation of the motion and the measurement. 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 and they remain unchanged within a pulse. This is different from [2] since our method does not take the velocity and acceleration values as the components in the state vector. Before the absolute velocity value is high enough, equation (4) is used for prediction. When the velocity reaches certain level, equation (3) is used instead. However, the preliminary values of $K_1$ and $K_2$ should be determined first in order to update the state vector in 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 value as shown in the previous section. However, zero cannot be used to evaluate the preliminary values of $K_1$ and $K_2$.) So, we have:

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

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

By solving equations (5) and (6), we obtain

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

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

where $t$ will be incremented by 1 as we begin to predict the next velocity value. The iteration repeats until we have predicted four consecutive velocity values, which are then added up to become the predicted mouse position for the next step. If $v_1$ or $v_2$ taken is not valid (say, one of them equals to zero), the system will continue to use equation (4) for prediction and both $v_1$ and $v_2$ taken previously are ignored. Sometimes, $t$ may reach $2\pi/K_1$. If this happens, the predictor will terminate the pulse and wait for the start of a new pulse.

## 5. Results and Discussions

We have implemented four prediction methods. They are:

- *Method 1* always predicts the next mouse velocity as zero.
- *Method 2* is new method proposed here.
- *Method 3* is a 2nd order polynomial predictor with Kalman filter [2].
- *Method 4* is the EWMA with residual adjustment (EWMA-R) [8].

We have studied the performances of these methods with and without navigation constraints. We conducted five sets of experiments. Each corresponds to a different walking pattern. The experiments are conducted on our prototype application implemented using OpenInventor. Figure 3 shows a view of the virtual environment used in our test. It is obtained from a well-known game called DOOM [21].

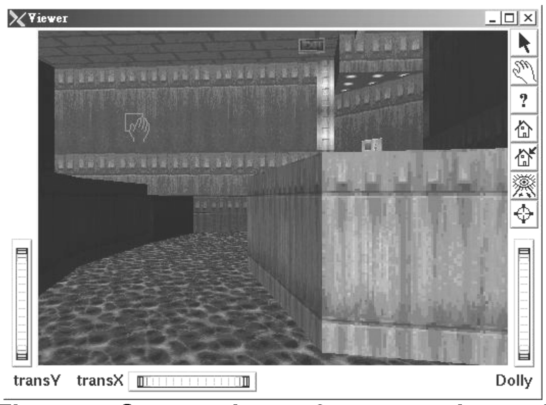

**Figure 3. Screen shots of our experimental prototype.**

Figure 4 shows the walking pattern we used to test the four prediction methods. In experiment #1, we experimented the CP pattern, which models a constant circular translation pattern. The user moves circularly starting and ending at the same location. In experiment #2, we experimented the CCP pattern which models the same pattern as CP except that the moving direction changes less rapidly at each step. The remaining three experiments perform the random walk (RW) with various degrees of vigor. The more vigorous the random walking pattern is, the more rapid is the mouse motion. The degree of vigor is increased gradually from experiment #3 to experiment #5. All the experiments are performed on a PC with a Pentium III 800MHz CPU and 128MB RDRAM. The results are presented in the following figures and tables.

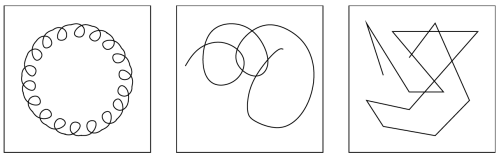
**Figure 4. Moving patterns: (a) CP, (b) CCP, and (c) RW.**

From tables 1 - 4, we may observe that EWMA-R is not a good choice comparing with other methods when a 2D mouse is used as an input device. The method predicts 3D future movements only from past 3D movements and ignores the fact that all movements in the 3D environment are induced by a 2D mouse. The other three prediction methods, however, are based on predicting the mouse motion. The experimental results clearly indicate that the EWMA-R method produced a much higher prediction error than the other three methods.

**Table 1. Errors in predicting angular velocity without navigation constraints.**

| | exp #1 | exp #2 | exp #3 | exp #4 | exp #5 |
|---|---|---|---|---|---|
| method 1 | 0.0161 | 0.00892 | 0.0183 | 0.0312 | 0.0423 |
| method 2 | 0.0177 | 0.00890 | 0.0173 | 0.0292 | 0.0653 |
| method 3 | 0.0306 | 0.0145 | 0.0281 | 0.0346 | 0.0574 |
| method 4 | 1.18 | 1.15 | 0.866 | 0.887 | 0.737 |

**Table 2. Errors in predicting translation velocity without navigation constraints.**

| | exp #1 | exp #2 | exp #3 | exp #4 | exp #5 |
|---|---|---|---|---|---|
| method 1 | 4.63 | 3.51 | 29.8 | 38.3 | 47.6 |
| method 2 | 4.44 | 2.93 | 27.6 | 32.8 | 50.2 |
| method 3 | 12.8 | 10.2 | 50.5 | 43.7 | 62.4 |
| method 4 | 20.5 | 31.7 | 65.2 | 83.9 | 84.5 |

**Table 3. Errors in predicting angular velocity with navigation constraints.**

| | exp #1 | exp #2 | exp #3 | exp #4 | exp #5 |
|---|---|---|---|---|---|
| method 1 | 0.00799 | 0.00521 | 0.00946 | 0.0138 | 0.0150 |
| method 2 | 0.00620 | 0.00406 | 0.00642 | 0.00977 | 0.0104 |
| method 3 | 0.0108 | 0.007 | 0.00905 | 0.0104 | 0.0118 |
| method 4 | 1.21 | 1.15 | 0.859 | 0.872 | 0.736 |

**Table 4. Errors in predicting translation velocity with navigation constraints.**

| | exp #1 | exp #2 | exp #3 | exp #4 | exp #5 |
|---|---|---|---|---|---|
| method 1 | 3.52 | 2.90 | 16.3 | 18.7 | 20.7 |
| method 2 | 2.77 | 2.06 | 11.4 | 12.5 | 14.6 |
| method 3 | 6.94 | 5.60 | 16.6 | 14.5 | 17.2 |
| method 4 | 20.8 | 31.9 | 63.9 | 81.6 | 81.9 |

Figures 5 - 8 show the same results as tables 1 - 4 but in the form of bar charts. In order to compare the performances of the other three methods, we have not included the results of the EWMA-R method due to its significantly higher prediction error. Figure 9 shows the average mouse motion speed, which is proportional to the degree of vigor of the motion. From figures 5 to 8, we can see that in general, the prediction error increases proportionally with the average mouse motion speed. Roughly, a higher speed may imply that the walking pattern contains a higher frequency and thus causes a higher error [2]. Another observation is that the constrained navigation algorithm reduces the prediction error significantly. It is because the reduction in the degree of the freedom eases the prediction as described in Section 4. It may also be regarded as a low pass operation, filtering the mapping of the high frequency portion of the walking pattern to the actual 3D movement.

As shown in figures 5 and 6, the performance of method 1 is good without navigation constraints, given the simplicity of the method. This is mainly due to the fact that during a 3D navigation with a mouse, the user's hand does not move most of the time. We may also observe that our method (method 2) is, in general, more accurate than method 3, in particular when the average mouse motion speed is low. This indicates that our method performs better in low and moderate frequency motions.

Figures 7 and 8 show a dramatic improvement in performance for all methods. However, our method is now better than all the other methods in all the experiments. This is because the navigation constraints filter the high frequency portion of the motion, in experiment #5 in particular.

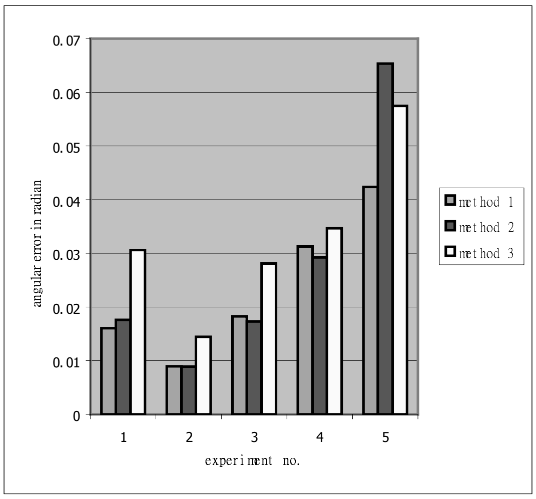

**Figure 5. Errors in predicting angular velocity without navigation constraints.**

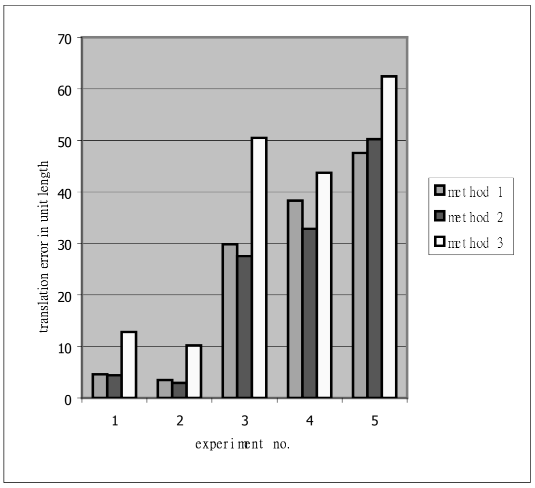

**Figure 6. Errors in predicting translation velocity without navigation constraints.**

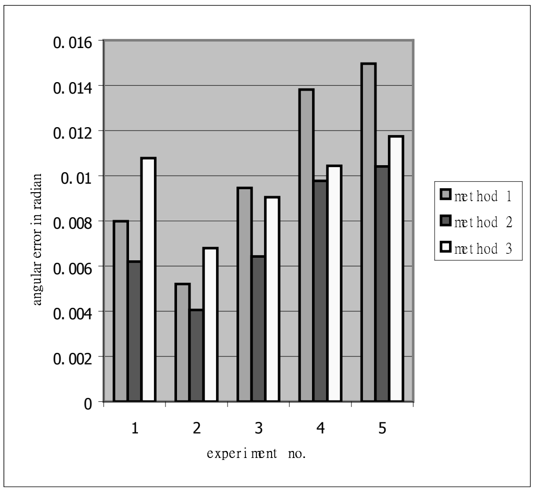

**Figure 7. Errors in predicting angular velocity with navigation constraints.**

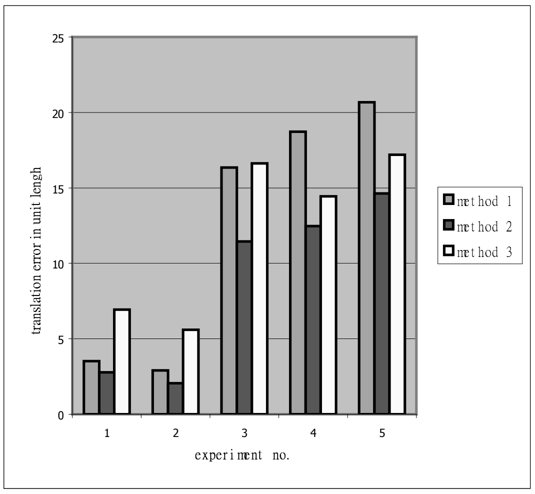

**Figure 8. Errors in predicting translation velocity with navigation constraints.**

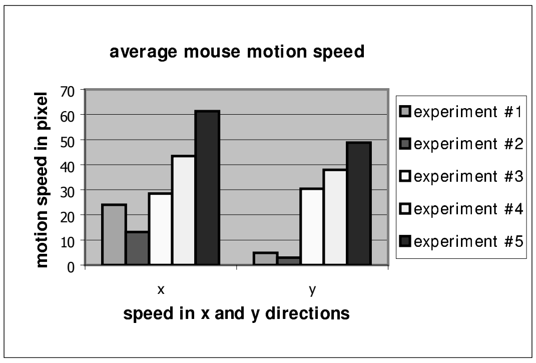

**Figure 9. Average speed of the mouse motion in x and y directions for each experiment.**

Table 5 compares the computational costs of the four methods. It is obvious that method 1 should be the fastest of all. Method 4 is also very efficient because of its simple computation. However, it may be surprising to see that our method is faster than the polynomial predictor with Kalman filter (method 3) even though our method also uses Kalman filter to estimate the values of the parameters, $K_1$ and $K_2$. The reason is that the state vector used in the method 3 have a size of 3, while those in our method only has a size of 2. Besides the length of the measurement vector in method 3 is 2. It is greater than the one in our method, which is 1 only because we only need the velocity value to update our estimation on $K_1$ and $K_2$ in the Kalman filter.

**Table 5. Computational cost for each prediction method.**

| | method 1 | method 2 | method 3 | method 4 |
|---|---|---|---|---|
| computation time (in ms) | 0.01 | 0.18 | 0.6 | 0.029 |

## 6. Conclusions

In this paper, we have proposed a motion prediction method for predicting the motion of a 2D mouse during 3D navigation. We have demonstrated the effectiveness of the new method compared with other popular methods through experiments. We are currently porting the method to the distributed virtual walkthrough system that we have developed. Extensive experiments will then need to be conducted to find out the impact of the proposed method on the hit ratio of the prefetching mechanism and the overall system latency. In addition, we will also look at how the proposed method may benefit real applications such as flight simulator and collaborative DVE.

## 7. Acknowledgements

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).

## 8. References

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

[2] R. Azuma and G. Bishop. "A Frequency-Domain Analysis of Head-Motion Prediction." In *Proceedings 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: Teleoperators and Virtual Environments*, **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 *Proceedings of IEEE VRAIS*, pages 450-455, 1993.

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

[7] 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 *Proceedings of ACM Multimedia*, pages 171-180, 1998.

[8] 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 *Proceedings of ACM VRST*, pages 25-34, 1998.

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

[10] 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.

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

[12] A. Hanson and E. Wernert. "Constrained 3D Navigation with 2D Controllers." In *Proceedings of IEEE Visualization '97*, pages 175-182, 1997.

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

[14] 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.

[15] J. Mackinlay, S. Card, and G. Robertson. "Rapid Controlled Movement Through a Vitrual 3D Workspace." In *Proceedings of ACM SIGGRAPH'90*, pages 171-176, 1990.

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

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

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

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

[20] The VRML Specification. Available at http://www.web3d.org/

[21] Wadtoiv. Available at http://www-white.media. mit.edu/~kbrussel/wadtoiv.html