

# On Temporal-Spatial Realism in the Virtual Reality Environment

Jiandong Liang, Chris Shaw and Mark Green  
Department of Computing Science,  
University of Alberta,  
Edmonton, Alberta, Canada, T6G 2H1  
{leung,cdshaw,mark}@cs.ualberta.ca

## Abstract

The Polhemus Isotrak is often used as an orientation and position tracking device in virtual reality environments. When it is used to dynamically determine the user's viewpoint and line of sight ( e.g. in the case of a head mounted display) the noise and delay in its measurement data causes temporal-spatial distortion, perceived by the user as jittering of images and lag between head movement and visual feedback. To tackle this problem, we first examined the major cause of the distortion, and found that the lag felt by the user is mainly due to the delay in orientation data, and the jittering of images is caused mostly by the noise in position data. Based on these observations, a predictive Kalman filter was designed to compensate for the delay in orientation data, and an anisotropic low pass filter was devised to reduce the noise in position data. The effectiveness and limitations of both approaches were then studied, and the results shown to be satisfactory.

## 1 Introduction

In recent years, the virtual reality concept has been explored by many researchers [Krueger83] [Fisher86] [Brooks86] [Wang90] [Green90]. To provide real-time visualization of a 3-D space, a Polhemus Isotrak tracker is often used to determine the user's viewpoint and line of sight, and based on the measured data, stereoscopic images are generated and displayed to the user through the head mounted display. In this paradigm, two factors adversely affect the user's perception of the virtual world, namely, the jittering of images and the lag between head movement and visual feedback [Rebo89] [Wang90].

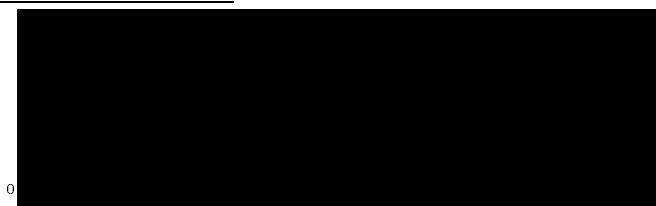
The jittering of images is caused by the noise in the measured data. The lag is due partly to the computation and rendering time needed to generate the images, and partly to the delay in the measured data. To generate new tracking data, the Isotrak generates and senses electromagnetic fields, calculates the sensor location, and sends the new location data to the host via a serial communications line. The measurement delay is the time taken by this three-step process.

Improving the hardware alone cannot solve the problem. First, the Isotrak is subject to electromagnetic interference, which is common in computer laboratories. Second, after the viewing position and orientation is available to the virtual reality software, a certain amount of computation is needed to manage interaction and generate images. Third, in an environment with multiple trackers (e.g. one for a head mounted display and one for each DataGlove), the devices must be time sliced to avoid interference amongst themselves, thus reducing the sampling rate for each tracker and increasing the delay (see Section 2 below).

In the following section, we discuss the measurement of the noise and delay. Section 3 focuses on the compensation for delays in orientation data by applying predictive Kalman filtering. Section 4 discusses how we can apply an anisotropic low pass filter to position data in order to reduce the perceived noise. Section 5 concludes the study and points out directions for future research.

## 2 Measurement of Noise and Delay

The noise in the Isotrak data can be measured by plotting deviations about the mean for a large set of samples. Figure 1 shows the deviations in both position and orientation for a source and sensor each fixed to the same stable platform at a relative distance of 20 inches. The peak-to-peak maximum noise level in the position data is about one order of magnitude higher than the noise in orientation data.



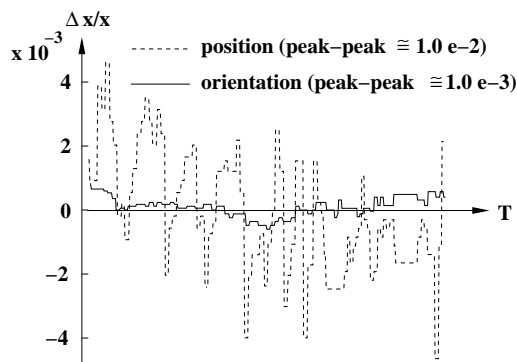


Figure 1. Noise during a 1-minute period.

## 2.1 Delay Measurement

Delay can be measured by using a reference tracking device to measure the lag between the actual sensor position and the position reported by the Isotrak. The sensor should be mounted on an object that moves in a known manner. The motion must be tracked by a reference tracking device that has the following properties: (i) the delay of the reference device must be known exactly, or should be small enough to be ignored compared with the delay of the Isotrak, (ii) the device must not cause electromagnetic interference with the Isotrak, and (iii) it must not bring any metallic object into close proximity with the Isotrak sensor or source. These three conditions can be met by using a video camera to track the periodic motion of a pendulum.

In the pendulum tracking experiment, the Isotrak sensor was attached to a non-metallic swinging pendulum (see Figure 2). The source was mounted on a stable reference pendulum, and the video camera was arranged so that the swinging and reference pendulums coincided when the swinging pendulum was at its neutral position. The computer displayed a time stamp on the screen each time a new set of Isotrak data was received, and stored the data with the time stamp in a log file. The video camera recorded the swing of the pendulum and the time stamp as it was displayed on the screen. Later, the video tape was played back frame by frame, and when the pendulum was aligned with the reference pendulum (i.e., in its neutral position), the corresponding time stamp was noted. In this way, we found the displacement of the pendulum as known by the computer when it was actually at the neutral position. This displacement can be easily converted to lag time, since the amplitude and period of the pendulum can both be extracted from the log file.

In order to measure only the lag induced by the Isotrak and the driver software, the time logger program was heavily optimized. No disk I/O was performed by the logger while the experiment was in progress, and logins by other users were banned. Most importantly, no screen clears were done before drawing a new number,

and no frame buffer swaps were performed. Thus, any added lag was due solely to collecting the latest time stamp, blanking out the old time stamp, and drawing the new one on the screen.

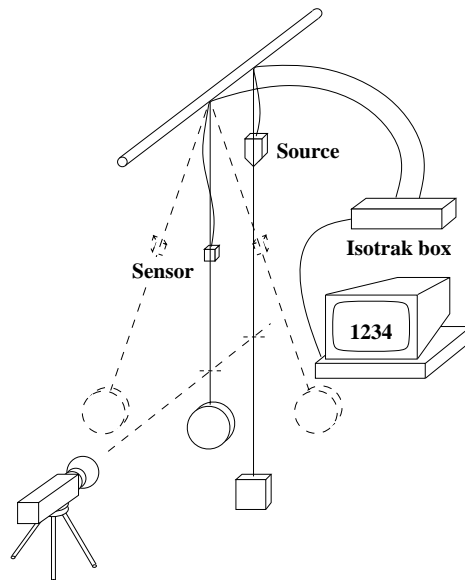


Figure 2. The pendulum experiment.

## 2.2 System Configuration

In our hardware configuration, we had three Polhemus 3Space Isotraks running in the same local space, synchronized at 20 Hz in a round-robin scheme. Our usual software configuration drives each Isotrak with its own server [Green90], which accepts client connection requests, and sends the latest Isotrak data in response to a client request. The client-server connection is across an Ethernet if the client is on a different machine than the server. The server keeps up-to-date by continuously polling the Isotrak for new data. The Isotrak used in this experiment was driven by an Iris 3130, and two client machines were used, the 3130, and an Iris 4D/20.

The time-of-day clock on the 3130 runs at 60 Hz, and the clock on the 4D/20 runs at 100 Hz. In order to find how much time is spent by time stamp collection and the data logging code of the logger program, the data collection call in the update loop was removed, and the loop was timed for 5000 iterations. This resulted in a time value of approximately 4 ms for both the 3130 and for the 4D/20. Thus, the “application code” used in this experiment took almost no time in comparison with the data collection process.

An alternate means of accessing the Isotrak is for the client software to bypass the server and do direct I/O with the appropriate serial port. The disadvantage is that only one process on the local machine has access to the Isotrak in this manner.

### 2.3 Experimental Results

The experiment was conducted for different configurations of the software and hardware, in order to determine the effect of each factor.

The hardware settings were: (i) sampling rate of the Isotrak box (20 Hz, or 60 Hz), and (ii) whether to communicate data across the network.

The software options were: (i) communication mode between the Isotrak and host machine (host machine polling Isotrak, or Isotrak continuously sending data to host machine), and (ii) access of data by client program (direct, or via client-server connection).

For each system configuration shown in Tables 1 and 2, three experiments were conducted, where each experiment recorded the swinging pendulum for 20 or more cycles. The delay number reported was the total lag, including measurement time, communications time, and drawing time.

It is obvious that the most important factor is the sample frequency of the Isotrak. Lag can be reduced by anywhere between 20 and 55 milliseconds by using a higher sampling frequency. The largest improvement was when the test program did direct access polling, and the smallest is when the server was obtaining continuous data.

Table 1. Delay in polling mode (ms).

Isotrak frequency	direct access	server	
		remote	local
60 Hz	85	130	110
20 Hz	140	180	160

Table 2. Delay in continuous mode (ms).

Isotrak frequency	server	
	remote	local
60 Hz	105	90
20 Hz	130	110

The second most important factor is the communications mode between host and Isotrak. Sending continuous data from the Isotrak improved lags by 20 to 50 ms. In fact, the second fastest result is where continuous mode was used to serve a local client, at a lag of 90 ms (Table 2).

The third factor to consider is direct I/O versus client-server communication. For polling mode, the improvement gained by direct I/O is 20 to 25 ms, but this benefit can only be realized on the host machine. Also, the lag of 85 ms was strictly best-case, since the “application” was almost nonexistent.

In order to achieve this lag in practical situations, the poll request must be carefully placed in the application code to produce the most up-to-date data just before the program needs it. In the logger program, properly placing the poll request is easy due to the shortness (about 4 ms) of the update loop.

In a real application, poll request placement is more difficult, since if the poll request occurs too late, then the data will not arrive in time, forcing the application to wait, or worse, forcing the use of old data. If the poll request is too early, then needless extra time is spent, and the application runs the risk of using data that is old by one Isotrak sample time. The programmer may attempt to reduce lag by placing multiple poll requests throughout the application code, but this means that the application must be precisely timed to avoid useless poll requests.

In continuous mode, the problems of using direct access are compounded by the real time problems of managing an input queue that is always filling up. This queue management contributes directly to the application’s update latency, and it must be done correctly to avoid getting old data. The upshot is that direct I/O has significant costs in terms of software engineering, and it is not clear that the 20-25 ms improvement can ever be achieved in practice.

Lastly, the effect of the network setting is quite small, adding between 15 and 20 ms for cross-network communications. However, these experiments were conducted when our network was assumed to be lightly loaded. In situations of higher network load, greater delay may occur.

### 3 Orientation Data

The delay in orientation data accounts for most of the lag felt by the user, primarily because the user changes viewing direction more frequently than position due to the limitation on Isotrak source-sensor distance (about 60 inches). Also, the change of viewing direction often causes more noticeable changes in the scene. In other words, if the delay in orientation data can be partially compensated for, then the user will feel less lag, even though the delay in position data remains the same.

However, compensating for this delay is far from trivial. Experiments showed the inadequacy of simple extrapolation schemes. There are two major reasons for this. First, the user’s head movement can not be predicted, nor can it be depicted by a deterministic model. Second, the orientation data is noisy (though it is ‘cleaner’ than the position data).

Fortunately, the Kalman filtering technique [Brown83] is quite suitable for this situation, since it accommodates both the stochastic nature of a process and the error (noise) in the measured data.

Due to space limitations, the details of Kalman filtering cannot be presented in this paper. For an introduction to the theory and practice behind this filtering technique, see any of the standard signal processing texts, such as Brown [Brown83].

To apply this technique, two problems must be solved: the linearization of representation, and a proper choice of random process model.

### 3.1 Linearization

There are many ways to represent orientation in 3-D space, e.g. Euler angles, rotation matrix, and unit quaternion. For computer graphics, the unit quaternion is often the best choice, due to its simplicity, continuity at all attitudes, and immunity from gimbal lock [Shoemaker85]. Unfortunately, a unit quaternion resides on the unit sphere in 4-D space, and does not lend itself to a linear representation as required by the Kalman filter.

To linearize the quaternion of the form  $q = (\cos \frac{\theta}{2}, x \sin \frac{\theta}{2}, y \sin \frac{\theta}{2}, z \sin \frac{\theta}{2})$ , the magnitude of rotation  $\theta$  can be extracted, leaving the unit vector  $n = (x, y, z)$  as the axis of rotation. Then, employing the assumption that the change of orientation in one sampling period is small, as used by the tracking mechanism itself [Raab79], we temporarily relax the unit length constraint, independently filter  $x, y$ , and  $z$ , and normalize the filtered  $x, y, z$  back to a unit vector  $n'$ , which is then coupled with the filtered  $\theta$  to form the resulting quaternion.

### 3.2 Head Movement Model

User head motion is usually characterized by bursts of changes in viewing direction followed by relatively long periods of static viewing direction. Thus, the underlying assumptions are as follows: 1) The user's change of viewing direction is infrequent. 2) The angular speed and the angular acceleration are nonzero only during the infrequent changes in orientation. The angular speed and acceleration correspond to the frequency of the orientation "signal". In this situation, we model the user's head orientation as a random variable which changes at random times, and has limited rate of change during these times. Based on this analysis, we chose an integrated Gauss-Markov process<sup>1</sup> to model the head movement.

The state equation has the following form:

$$x'' = -\beta x' + \sqrt{2\sigma^2\beta}w(t), \quad (1)$$

where  $x', x''$  are the first and second derivative of the variable we are filtering,  $w(t)$  is a unit white noise sequence,  $\beta$  is a time factor, and  $\sigma^2$  is a variance factor. This model reflects our underlying assumptions that 1)  $x'$  tends to be zero (due to the negative coefficient  $-\beta$ ), i.e.  $x$  tends to remain static, 2)  $x''$  is driven by a random noise, which accounts for the acceleration during

<sup>1</sup> A random process  $X(t)$  is called a *Gauss-Markov* process if it has an exponential autocorrelation, and all the density functions describing the process are normal in form and invariant under a translation of time. The autocorrelation and spectral functions are of the form:

$$R_X(\tau) = \sigma^2 e^{-\beta|\tau|}, \quad \text{and} \quad S_X(j\omega) = \frac{2\sigma^2\beta}{\omega^2 + \beta^2}.$$

bursts of rotational movements, and 3) there is a relatively stable time constant  $\beta$ , which reflects the limit of speed and acceleration of head movements.

By adjusting the time factor  $\beta$  and variance factor  $\sigma^2$ , we can control the smoothness of the prediction and lessen the overshoot. Greater values of  $\beta$  increase the rate at which velocity is predicted to return to zero, at the expense of overshoot.

A second state equation relates the measured data  $y$  with its actual value  $x$ :

$$y = x + \gamma v(t), \quad (2)$$

where  $v(t)$  is the noise in the measured data. Due to the lack of knowledge about the nature of the noise, we assume  $v(t)$  to be a unit white noise sequence uncorrelated with the  $w(t)$  sequence.  $\gamma$  is the magnitude of the noise. In our environment,  $\gamma \approx 0.001$ , as was shown in Figure 1.

We attribute the problem of excessive overshoot [Wang90] and oscillation [Rebo89] encountered by other researchers to an inappropriate choice of model.

For example, Rebo and Amburn [Rebo89] use a simple model which assumes that the acceleration is a random number between  $-1.0$  and  $1.0$ . This implies that the velocity is a random walk, and has a variance linearly increasing with time, which does not reflect the nature of head movement (users do not tend to move wilder as time progresses).

### 3.3 Prediction Length

Given the assumption that the noise in the measured data is both white and at a low amplitude, and that the Nyquist limit of head motion is about 10 Hz, this means that a 20 Hz sample rate is an adequate sampling frequency for prediction. In our system, the Isotraks are synchronized at a 20 Hz sampling frequency, yielding a 50 ms sampling step size. This step size will be a basis for the discussion of the following quantitative results.

### 3.4 Quantitative Results

A Kalman filter performs best when it predicts one step ahead of its collected measurement data. From section 2 we see that in continuous mode at 20 Hz, we get a lag of 110 ms, so we would like to predict at least two steps ahead. However, the reliability of the prediction decreases as the length of prediction increases. Figure 3 shows the effect of a 3-step prediction. The dashed line shows the measured data, which lags 150 ms behind the actual movement (shown as dotted line). The solid line shows the output produced by a 3-step predictive Kalman filter. In this case, the Kalman filter compensates for the lag fairly well, as the solid line almost coincides with the dotted line.

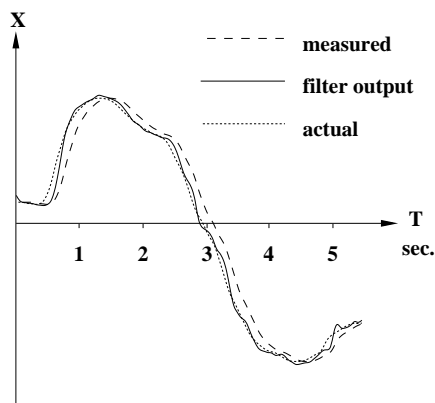


Figure 3. Predicting 3 steps ahead.

By contrast, figure 4 shows the effect of a 10-step prediction. In this case, the measured data is 500 ms behind the actual data, and the Kalman filter is set up to compensate for this 500 ms lag. As can be seen, the Kalman filter fails to compensate for all the lag, and the output exhibits extra zigzags (noise). Overall, the longer the lag that Kalman filter tries to compensate for, the less accurate and more noisy the output is.

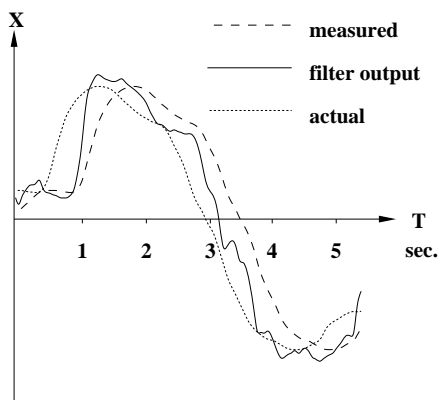


Figure 4. Predicting 10 steps ahead.

Figure 5 shows the relationship between the amount of delay the Kalman filter compensates for and the length of prediction. As the length of prediction increases, the absolute amount of compensation first increases to a maximum of about 185 ms, then slowly decreases. That is, lag compensation cannot exceed 185 ms, no matter how many steps of Kalman prediction performed. In fact, predicting more than 10 steps worsens the lag compensation performance.

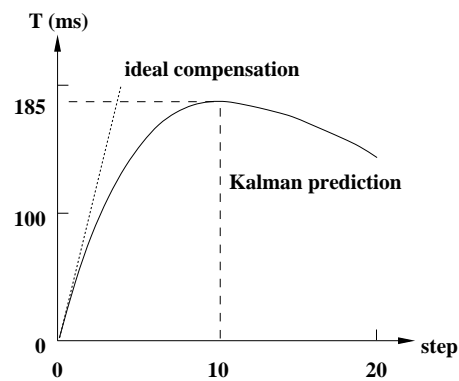


Figure 5. Lag compensation vs. length of prediction .

However, this figure only reflects the temporal aspect of the output. It does not reflect the spatial fidelity (accuracy) of the output. Although we do not have a good yardstick for measuring the overall “fidelity” or “accuracy” of prediction, we think the spectrum of the output signal partially captures this idea. Figure 6 shows the spectra of the orientation data for different prediction lengths. It demonstrates the fact that extra noise is added into the data for increased prediction lengths.

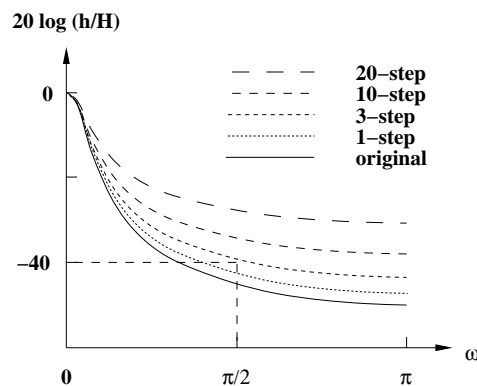


Figure 6. Spectra of orientation data.

From our experience, the maximum tolerable level of the high frequency component of orientation data is about  $-40$  dB. This limits the length of prediction to be within 3 steps, which is nominally 150 ms for a sampling rate of 20 Hz. In other words, the requirement for spatial fidelity puts this constraint on the amount of temporal compensation that can be made. A trade-off must be made between noise level and prediction length, and 3 step prediction seems to be the optimal choice.

### 3.5 Algorithm and Parameters

From equations 1 and 2, the filter algorithm for a single component is:

$$\begin{aligned}
 & \text{Kalman}(\beta, \sigma^2, \gamma, \delta, \tau) \\
 & \{ \\
 & \quad \mathbf{x} = \begin{pmatrix} z_0 \\ 0 \end{pmatrix}; \\
 & \quad H = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\delta} \end{pmatrix}; \\
 & \quad P = I; \\
 & \quad R = \gamma^2 H^T H; \\
 & \quad q_{11} = \frac{2\sigma^2}{\beta} [\delta - \frac{2}{\beta}(1 - e^{-\beta\delta}) + \frac{1}{2\beta}(1 - e^{-2\beta\delta})]; \\
 & \quad q_{12} = 2\sigma^2 [\frac{1}{\beta}(1 - e^{-\beta\delta}) - \frac{1}{2\beta}(1 - e^{-2\beta\delta})]; \\
 & \quad q_{21} = q_{12}; \\
 & \quad q_{22} = \sigma^2(1 - e^{-2\beta\delta}); \\
 & \quad Q = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{pmatrix}; \\
 & \quad \Phi = \begin{pmatrix} 1 & \frac{1}{\beta}(1 - e^{-\beta\delta}) \\ 0 & e^{-\beta\delta} \end{pmatrix}; \\
 & \quad \Psi = \begin{pmatrix} 1 & \frac{1}{\beta}(1 - e^{-\beta\tau}) \\ 0 & e^{-\beta\tau} \end{pmatrix}; \\
 & \quad \text{for } i = 1 \text{ to } n \{ \\
 & \quad \quad K = PH^T(HPH^T + R)^{-1}; \\
 & \quad \quad \mathbf{z} = \begin{pmatrix} z_i - z_{i-1} \\ z_i \end{pmatrix}; \\
 & \quad \quad \mathbf{x} = \mathbf{x} + K(\mathbf{z} - H\mathbf{x}); \\
 & \quad \quad P = \Phi(I - KH)P\Phi^T + Q; \\
 & \quad \quad \begin{pmatrix} y_i \\ y'_i \end{pmatrix} = \Psi\mathbf{x}; \\
 & \quad \quad \text{output}(y_i); \\
 & \quad \quad \mathbf{x} = \Phi\mathbf{x}; \\
 & \quad \} \\
 & \}
 \end{aligned}$$

where  $z_0, z_1, \dots, z_n$  are the measured data,  $\delta$  is the length of a step,  $\tau$  is the length of prediction, and  $y_i$  gives the prediction based on  $z_0, \dots, z_i$ . In practice, the variables  $H$ ,  $R$ ,  $Q$ ,  $\Phi$ , and  $\Psi$  are constant, since the Kalman parameters  $\beta$ ,  $\sigma^2$ ,  $\gamma$ ,  $\delta$ , and  $\tau$  are each fixed at some optimal value. The variables  $P$  and  $\mathbf{x}$  maintain the state of the component being filtered. There are four components in our application, so four sets of state variables are used. At each new sample, the body of the for loop is executed once for each component, using the state variable built up by filtering the previous samples.

To find the optimal Kalman parameters, suppose  $\mathbf{q}_i = (q_i^{(0)}, q_i^{(1)}, q_i^{(2)}, q_i^{(3)})$ ,  $i = 0, \dots, n$  is the sequence of measured quaternions, and  $\mathbf{p}_i = (p_i^{(0)}, p_i^{(1)}, p_i^{(2)}, p_i^{(3)})$ ,  $i = 1, \dots, n$  is the sequence we obtain by predictive Kalman filter of  $k$  steps, i.e.  $\tau = k\delta$ . We define the difference between the predictions and actual values to be

$$D_k(\beta, \sigma^2) = \sum_{j=1}^{n-k} \sum_{l=0}^3 |p_j^{(l)} - q_{j+k}^{(l)}|. \quad (3)$$

Using a hill climbing algorithm, we found a set of parameters  $\beta, \sigma^2$  which minimized  $D_k$  for  $k = 1, 2, \dots$ . After the hill climbing algorithm was applied *a posteriori* to a large set of test runs, the results were averaged to yield the best values for *a priori* (practical) use. For example, the best values for  $\delta = 50$  ms, and  $\tau = 150$  ms are  $\beta \approx 8.7$  and  $\sigma^2 \approx 0.2$ . To complete the parameterization, the measured noise magnitude  $\gamma \approx 0.001$  was used, as shown in Figure 1.

### 4 Position Data

The jittering of images perceived by the user is primarily due to the noise in position data. This is because the noise level in position data is significantly higher than that in the orientation data, as was shown in Figure 1.

Experiments showed that a low pass filter of order 8 or higher is needed for reducing the noise to a tolerable level, which is about  $-60$  dB from our experience. Figure 7 shows the noise reduction produced by moving  $k$ -average low pass filters ( $k = 1, 4, 8, 12$ ).

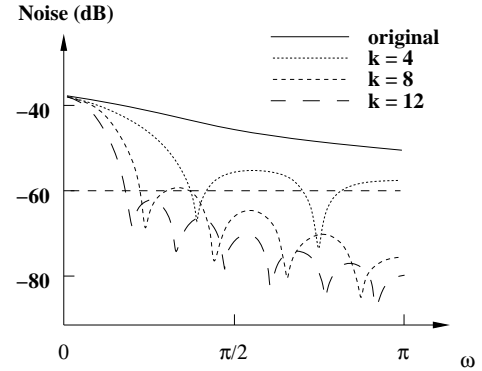


Figure 7. Noise reduction rate for low pass filters of various orders.

The additional lag produced by this type of low-pass filter is too large for our application. For example, when the sampling rate is 20 Hz and the filter order is 8, the additional delay is 400 ms.

#### 4.1 Observations

Again, we can make use of the characteristics of user movement and human perception under this specific situation.

First, we observed that the eye's sensitivity to the jittering of the scene is not uniform along all directions. Instead, the jittering perpendicular to the line of sight is much more noticeable than that along the line of sight, since for equal amounts of displacement, the former causes a more significant change of the perspective scene. This fact leads to a different tolerance of jittering along different directions.

Second, we noticed that most of the lateral (translational) part of the user's head movements are along the line of sight. This can be partially attributed to the evolution of the human body, since we usually move along the direction we are looking. Thus, the translational lag along the line of sight is of more significance to the user's perception. In other words, lag along the line of sight and noise perpendicular to the line of sight must be minimized.

## 4.2 Anisotropic Low Pass Filter

Based on these two observations, we devised an anisotropic filtering scheme. The idea is to use filters of different order along different directions, i.e.

- use a high order (about 8 – 10) filter for translational change perpendicular to the line of sight, in order to achieve a high rate of noise reduction, and
- use a low order (about 2 – 3) filter along the line of sight to keep a low overall additional delay.

To implement the anisotropic filter, suppose  $M$  is the rotation matrix from world coordinates to viewing coordinates. We first rotate the sequence of eye positions by  $M$  so they are in viewing coordinates<sup>2</sup>, and then filter each direction by their corresponding low pass filters. The result is then rotated back to world coordinates by  $M^{-1}$ .

Overall, the additional delay is about 50 to 75 ms, and the noise is reduced to below  $-60$  dB. The user experiences longer delay when he moves sideways, and shorter delay when he moves along the line of sight. While it is unusual, the visual effect is satisfactory.

## 5 Conclusions

The problem of improving the temporal-spatial realism in virtual reality environments was studied. We first measured the noise and delay in the Isotrak data, and identified the major causes of the user's perception of jittering and lag in the images. While the lag sources can be identified, they cannot be eliminated, so a predictive Kalman filter was designed to compensate for the lag in the orientation data. An anisotropic filtering scheme was devised to reduce the noise in position data while maintaining a low additional delay. Although the discussion is based on the Polhemus Isotrak as the tracking device, it is also valid for other head position/orientation tracking device in general.

However, many problems remain.

First, we need a yardstick for measuring the overall "fidelity" or "quality" of prediction.

Second, the filtering methods used here are not suitable for hand motion, since it does not possess many

of the characteristics we observed for head movement. There are fewer restrictions on hand movement, and the speed/acceleration of the motions is larger. We have observed that hand-eye coordination becomes very difficult when the display is slower than 4 updates per second. Even for a medium update rate of 10 updates per second, the lag between hand movement and visual feedback still makes certain tasks difficult, such as catching a moving object in the environment.

Third, it is not clear how to filter in a nonlinear domain such as 3-D orientation. Our linearization step is intended to sidestep this problem, but it would probably be better to use a filter scheme that works in the 4-D spherical domain.

## References

- [Brooks86] F. P. Brooks, Jr., "Walkthrough — A dynamic graphics system for simulating virtual buildings," *Proc. 1986 Workshop on Interactive 3D Graphics*, Chapel Hill, NC, October 1986, pp. 9–21.
- [Brown83] R. G. Brown, *Introduction to Random Signal Analysis and Kalman Filtering*, John Wiley & Sons, Inc. New York, 1983.
- [Fisher86] S. S. Fisher, M. McGreevy, J. Humphries, and W. Robinett, "Virtual environment display system," *Proc. 1986 Workshop on Interactive 3D Graphics*, Chapel Hill, NC, October 1986, pp. 77–87.
- [Green90] M. Green and C. Shaw "The DataPaper: Living in the Virtual World," *Graphics Interface 1990*, Halifax, Nova Scotia, May 1990, pp. 123–130.
- [Krueger83] M. W. Krueger, *Artificial Reality*, Addison Wesley, Reading, MA, 1983.
- [Raab79] F. H. Raab, E. B. Blood, T. O. Steiner, and H. R. Jones, "Magnetic position and orientation tracking system," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. AES-15, No. 5 (Sept. 1979), pp. 709–718.
- [Rebo89] R. K. Rebo and P. Amburn, "A helmet-mounted virtual environment display system," *Proc. SPIE*, Vol. 1116, pp. 80–84, 1989.
- [Shoemaker85] K. Shoemaker, "Animating rotation with quaternion curves," *ACM Computer Graphics*, Vol. 19 (1985), pp. 245–254.
- [Wang90] C. P. Wang, L. Koved, and S. Dukach, "Design for interactive performance in a virtual laboratory," *Proc. 1990 Symp. on Interactive 3D Graphics*, Utah, March 1990, pp. 39–40.

---

<sup>2</sup>More accurately, it's the viewing coordinates without the translation component.