Optimizing Avatar Environmental Update in Shared Virtual Reality Environments

James Kempf Ajay Chander Manhee Jo
NTT DoCoMo USA Labs
3240 Hillview Avenue
Palo Alto, CA 94304, USA
1-650-493-9600
{kempf, chander, mjo}@docomolabs-usa.com

ABSTRACT
A major problem with the current generation of shared 3D virtual reality game-like environments on the Internet is that update of the virtual environment during avatar movement often lags considerably behind the scene which the avatar's user perceives. The result is an unsatisfying, artificial experience, in which the user must halt avatar movement until the new scene slowly assembles before the avatar and sounds pick up, or allow the avatar to continue moving and risk bumping into objects that are revealed later when the view fully assembles. This paper describes an algorithm for optimizing environmental update based on a caching strategy for prefetching environmental data around a stationary avatar and a statistical model predicting avatar movement that is used for determining when and where to start updating the environmental data when the avatar starts moving.

Keywords
Virtual reality, graphics update, multiplayer on-line games.

1. INTRODUCTION
The current generation of shared virtual reality environments and massively multiplayer on-line role-playing games (MMORPGs) on the Internet depend on a central server farm to maintain the virtual reality environment shared by multiple users. The users interact within the environment through avatars which are rendered along with the shared environment on the users' remote hosts. One major problem with such virtual reality environments is that the update on a user's machine of the shared virtual graphics environment often lags considerably behind the actual movement of the user's avatar. This is particularly a problem when the avatar is moving quickly, such as when it is flying or is in a vehicle. The resulting visual experience is unsatisfying. The user is forced to either stop avatar movement while the newly revealed scene in front of the avatar assembles or continue moving while risking having the avatar bump into objects which are only revealed once the scene has fully assembled.

In this paper, we describe an algorithm for optimizing virtual environment update based on a caching strategy for stationary avatars and an algorithm for determining when and where to update during avatar movement. The caching strategy uses information about the avatar's maximum potential movement velocity when it begins moving and the maximum extent of the virtual environment it can see to calculate a prefetched region in which environmental data is cached on the client side to avoid network access for environmental data during the initial phase of avatar movement. When the avatar does start moving, an algorithm based on a statistical model called a Kalman filter [1] is used in order to predict where and when the avatar will outrun the prefetched data, and thereby when to schedule cache update. The Kalman filter has been widely used in automatic control for many years; most notably, it was incorporated into the navigation system of the Apollo command module for navigation to the moon, and it is also used quite widely in computer graphics applications [2]. The algorithms described in this paper operate independently from the server and can be implemented strictly on the client, though it is possible to optimize certain operations with server support.

In the next section, we briefly formulate the problem posed by avatar movement. Section 3 derives an estimate of the size for a cached area that would avoid having to update the virtual environment as soon as the avatar starts moving. Section 4, discusses using a Kalman filter to predict avatar movement. In Section 5, we describe how to estimate the time and place where an avatar will overrun the cached area, providing an indication for where and when to start prefetching environmental data for the avatar's eventual arrival, and how to microschedule updates of the graphical environment to minimize the consequences when subsequent movement by the avatar invalidates the prediction. Section 6 briefly examines previous work on the problem. Finally, Section 7 concludes the paper with some suggestions for future work.

2. PROBLEM STATEMENT
At any point in time, there are two disc shaped portions of the virtual environment centered on a stationary avatar within which the avatar can potentially perceive visual and aural events. We call the limiting circles for these discs the visual and aural event horizons for the avatar. In a 3D virtual environment, the event horizons extend in the vertical direction, but for simplified presentation, we can neglect the vertical direction without loss of generality. Note that the ability of the avatar to perceive events within the event horizon is only potential. Visual objects beyond a certain point within the visual event horizon may be blocked by nearer objects (high buildings, trees, or landscape) and sounds from a source beyond a certain point within the aural event

IMMERSCOM 2007, October 10-12, 2007, Verona, Italy.

DOI 10.4108/ICST.IMMERSCOM2007.2243

horizon may be blocked by an intervening object. Also, at any point in time, the avatar can only see visual events that are positioned in a pie shaped slice within its angle of view in front of it, the avatar's visual field of view (FoV), though aural events originating from the entire disc are perceptible unless blocked.

As shown in Figure 1, the aural event horizon is typically nearer than the visual event horizon, for an avatar that has the same sensory capabilities as a person1. This corresponds to the fact that the radius of heard events is considerably less than that of seen events. Since the visual event horizon covers a larger area and maintaining the visual scene typically involves considerably more computational update, we confine further discussion to the visual event horizon, though all the calculations apply to the aural event horizon too. In Figure 1, we designate the radius of the stationary avatar's visual event horizon as r_s and the angle subtended by the avatar's FoV as 2θ.

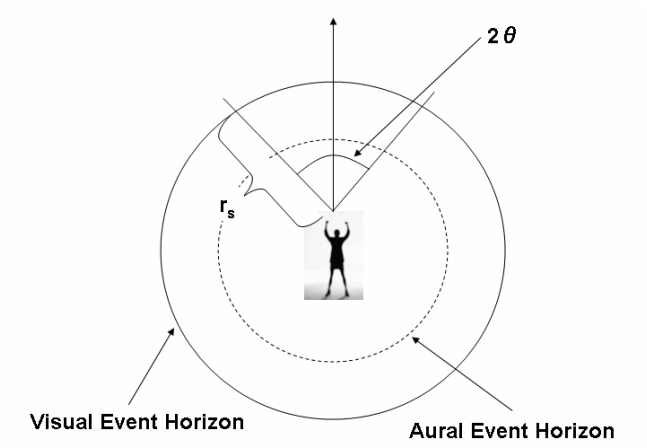

Figure 1: Avatar Event Horizons

Avatars can move by a variety of means depending on the virtual environment; walking, flying, or riding in a vehicle are a few examples. At any particular time, the avatar is likely to have a particular maximum velocity, V_{max}, at which it can move depending on the contrivances available. One strategy to avoid having to update the virtual environment over the network as soon as the avatar starts moving is to prefetch enough data outside the avatar's event horizon so that its event horizon can be smoothly moved without any need for network update over some period of time while the avatar is moving at its maximum velocity. This cached data will occupy an annular disc having the event horizon as its inner boundary and the extended event horizon as its outer boundary. Figure 2 illustrates. The radius of the extended event horizon is shown as r_e in the figure. A higher r_e corresponds to a larger cache size, and conversely. The point of intersection in polar co-ordinates between the avatar's movement vector and the extended event horizon is (r_i, ϕ_i). Our problem can be stated as follows: What is a lower bound estimate of r_e such that the user's experience of avatar movement is expected to be reasonably realistic?

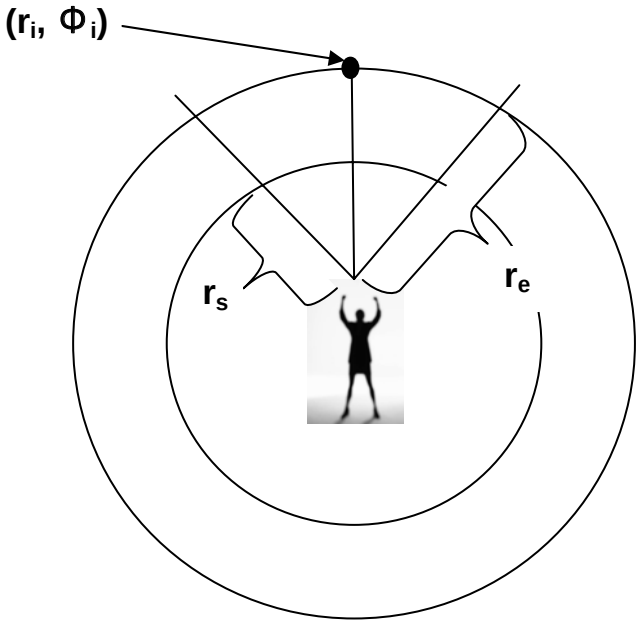

Figure 2: Extended Event Horizon

3. ESTIMATING THE EXTENDED EVENT HORIZON RADIUS

The time required to update the data within the extended event horizon depends on factors that are independent of the avatar's movement. Although object lookup on the server and caching on the client are likely to require some time, the bottleneck in obtaining the data is most likely to be the time required to transport the data from the server to the client, which we designate as t_N. This time can be estimated in the following way.

Suppose the current network throughput is T_b, in bytes per second. The amount of data required to be downloaded is the data density, D, in bytes per square meter of the local environment, times the area, A, in square meters, of the annular disc between the event horizon and the extended event horizon. The amount of time (in seconds) required to update the extended event horizon is:

$$t_N = \frac{DA}{T_b} \qquad \text{1)}$$

T_b and D can be estimated by the client. T_b is constantly changing depending on network conditions, but it is easy enough for the client to monitor by keeping track of its network performance. The client can estimate D by keeping track of how many bytes were downloaded to create the scene within the event horizon. This assumes that data density is somewhat uniform, and that a guess about average data density will suffice for general prediction purposes. Another way to estimate data density is to

[1] In some MMORPGs, avatars can use fictional advanced technology or magic to enhance their sensory capabilities; these enhancements need to be accounted for when calculating the event horizon.

Digital Object Identifier: 10.4108/ICST.IMMERSCOM2007.2243
http://dx.doi.org/10.4108/ICST.IMMERSCOM2007.2243

annotate the environmental objects with their size. This would allow the server to provide an exact estimate of the amount of downloaded data needed to construct a particular scene.

The only quantity on the right hand side of 1) that needs to be determined is A. From the geometry in Figure 2, the area of the annular disc between the extended event horizon and the event horizon is the difference between the area within the extended event horizon and the event horizon:

$$A = A_{EEH} - A_{EH} \quad \textbf{2)}$$

Using the formula for the area of a circle and the values of the radii for the extended event horizon and the event horizon results in the following:

$$A = \pi r_e^2 - \pi r_s^2 \quad \textbf{3)}$$

Substituting 3) into 1) results in:

$$t_N = \frac{(r_e^2 - r_s^2)\pi D}{T_b} \quad \textbf{4)}$$

We can estimate the time until the avatar's event horizon intersects the extended event horizon as follows. If the avatar's maximum available speed of movement is V_{max}, then the time until the avatar's event horizon intersects the extended event horizon is minimally:

$$t_I = \frac{r_e - r_s}{V_{\max}} \quad \textbf{5)}$$

In order to maintain the cache, the time until the avatar's event horizon intersects the extended event horizon, t_I in Equation 5, must be at least as long as the time necessary to download the data between the event horizon and the extended event horizon, t_N. If the download time is greater, then the avatar will experience a lag in the appearance of the environment. Since an avatar can choose to move in any direction, the data must be cached for the entire annular disk between the event horizon and extended event horizon. Thus, by equating equations 4) and 5), we can obtain an upper bound for r_e.

Equating 4) and 5) results in the following upper bound for r_e:

$$r_e \le r_s + \frac{1}{V_{\max} c} \quad \textbf{6)}$$

where the network and environmental parameters are gathered into c:

$$c = \frac{\pi D}{T_b} \quad \textbf{7)}$$

Note that as V_{max} and c increase, the value of r_e given by 6) approaches r_s. Intuitively, this means that a larger round trip time to the server, a more data dense environment, and a potential for a faster moving avatar will result in a smaller maintainable cached region.

In some cases, the values of these parameters may cause the size of the estimated precached region to become small enough that the avatar outruns the extended event horizon shortly after it begins moving. If that occurs, there are a couple of qualitative strategies that can be undertaken within the virtual environment in order to avoid having the avatar outrun the precached region:

1) Reduction of V_{max} by disallowing fast movement modes such as flying or by increasing the resistance to movement, such as oncoming waves in water or head wind in air. In general there may be limitations on how closely movement velocity can be controlled because avatars may have a built-in "right" to a particular movement velocity given their current set of capabilities and the ability of those capabilities to interact with the fixed virtual environment surrounding the avatar.
2) Reduction of r_s, decreasing the stationary avatar's perceptual event horizon. For example, the visual event horizon can be reduced by causing the local weather to change and become hazier, ultimately in the limit perhaps a driving rain storm. The aural event horizon can be reduced by adding more background noise, wind, traffic noise, etc. Of course, any change in visual or aural properties applied to one avatar must be applied to all within the same local area of the virtual environment, since the virtual environment is shared, so those avatars with user machines not experiencing reduction of the cached region size will also be affected.

In many cases, an avatar may be moving fast because the local environment doesn't hold her attention much, or because she is headed to a particular area. In the former case, certain enticements can be offered to the user, such as the chance for communication or "remote" action – anything that is related to utility points in the game/world. In the latter case, predictive knowledge of the destination area can provide directional guidance to the prefetching algorithm for the cache, yielding much better performance.

4. PREDICTION OF AVATAR MOVEMENT

Once the avatar starts moving, the event horizon moves out into the disc of cached environmental data and the client program begins rendering this data to provide the avatar with a smooth view as movement proceeds. At some point, the event horizon will intersect the extended event horizon and the client will run out of cached data to render. The client must begin caching sufficiently prior to this event such that environmental data within the avatar's FoV will be available when the intersection of the event horizon

Digital Object Identifier: 10.4108/ICST.IMMERSCOM2007.2243
http://dx.doi.org/10.4108/ICST.IMMERSCOM2007.2243

and the extended event horizon occurs. A predictor technique is used to estimate where and when the intersection event will take place, based on the avatar's current position, velocity, and acceleration.

While the avatar is moving, changes in direction and velocity made by the user as the avatar traverses the virtual world cause the avatar's movement behavior to be much like that of a maneuvering aircraft. We use a Kalman filter for a maneuvering vehicle [3] to estimate the avatar's position, velocity, and acceleration.

The state variables for the Kalman filter are the position, velocity, and acceleration in the x and y directions at time *t*, and the data density, *D*. The state variable vector at timestep *k* assuming the discretization time interval Δt is:

$$\vec{x}_k = \left(x_k, \dot{x}_k, \ddot{x}_k, y_k, \dot{y}_k, \ddot{y}_k, D_k\right) \quad \textbf{8)}$$

To estimate the value of D_k, we just use the value observed at time D_{k-1}. The other estimates follow from the Kalman filter estimator for a maneuvering vehicle [3], except in this case, we assume that the avatar is always maneuvering, since the user can always choose to vary the velocity to investigate something, etc. When the avatar's radial velocity reaches V_{max}, the acceleration becomes zero, because V_{max} is the limit. This is expressed as a change in the Kalman filter state transition matrix when the radial velocity exceeds V_{max}.

4.1 Kalman Predictor

The discrete Kalman filter predictor at timestep *k* is given by the following vector equation:

$$\vec{x}_k = F\vec{x}_{k-1} + G\vec{w}_{k-1} \quad \textbf{9)}$$

$$P_k = F\ P_{k-1}F^T + GQG^T \quad \textbf{10)}$$

Where *F* is the state transition matrix, *G* is the input distribution matrix, P_k is the covariance matrix at timestep *k*, and *Q* is the covariance matrix of the process noise.

$\vec{w}_k$ is the random input vector having the following structure:

$$\vec{w}_k = \left(w_1, w_2, w_3\right) \quad \textbf{11)}$$

Values for *F* are derived from the equations of motion and the constraint on maximum radial velocity.

The radial velocity can be calculated from the velocities in the x and y directions by differentiating the equation for converting to radial co-ordinates, $r_i = \left(x_i^2 + y_i^2\right)^{1/2}$. This results in:

$$\dot{r}_i = \frac{x_i\dot{x}_i + y_i\dot{y}_i}{\left(\dot{x}_i^2 + \dot{y}_i^2\right)^{1/2}} \quad \textbf{12)}$$

If $\dot{r} \le V_{\max}$, the state transition matrix F is:

$$F = \begin{pmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \textbf{13)}$$

and if $\dot{r} > V_{\max}$ the state transition matrix is:

$$F = \begin{pmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \textbf{14)}$$

The value for *G* is:

$$G = \begin{bmatrix} \frac{\Delta t^2}{4} & 0 & 0 \\ \frac{\Delta t}{2} & 0 & 0 \\ 1 & 0 & 0 \\ 0 & \frac{\Delta t^2}{4} & 0 \\ 0 & \frac{\Delta t}{2} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{15)}$$

Digital Object Identifier: 10.4108/ICST.IMMERSCOM2007.2243
http://dx.doi.org/10.4108/ICST.IMMERSCOM2007.2243

4.2 Kalman Corrector

The updated value of the covariance matrix and state variable vector can be calculated based on measurements of the avatar's position, velocity, and acceleration and on the data density using the Kalman filter update equations. Unlike the case for measurements of a maneuvering aircraft, the values for all three movement state variables can be measured because the movement is being generated within the client side program.

The Kalman gain matrix *K* at timestep *k* is calculated as:

$$K_k = P_k H^T \left(H P_k H^T + R \right)^{-1} \quad \textbf{16)}$$

H is the (7 x 7) identity matrix because we assume that we can measure all state variables, the position, velocity, acceleration, and data density, and *R* is the (7 x 7) matrix of measurement noise covariance.

Using the Kalman gain matrix, the updated state variable vector and updated covariance matrix are calculated:

$$\vec{x}'_k = \vec{x}_k + K_k \left(\vec{z}_k - H\vec{x}_k \right)$$
$$P'_k = \left(I - K_k H \right) P_k \quad \textbf{17)}$$

where $\vec{x}'_k$ and P'_k are the corrected state variable vector and the corrected co-variance matrix, *I* is the (7 x 7) identity matrix, and $\vec{z}_k$ is the vector of state variable measurements.

An initial value of the co-variance matrix P_0 is required. Since the statistics of the movement process are initially not known, we assume that the co-variance matrix is diagonal (i.e. there are no cross-correlations in the noise) and that the values of the variance for the noise on the diagonal is a very small value, ε. Alternatively, if statistics have been kept for this user in the past, a prior co-variance matrix can be used.

5. ESTIMATING THE INTERSECTION EVENT

With the predictor in hand, the time and place of the intersection between the avatar's event horizon and the extended event horizon can now be calculated. At time *k*, the predictor is run forward using the current co-variance matrix and state variable vector until the value of the radial distance from the starting point, r_i, exceeds r_e (derived from 6). On each step, the position estimate must be converted to polar co-ordinates:

$$r_i = (x_i^2 + y_i^2)^{1/2}$$
$$\phi_i = \arctan\left(\frac{y_i}{x_i} \right) \quad \textbf{18)}$$

Then, r_i must be compared with r_e. The estimated time of intersection then is just:

$$t_i = i\Delta t \quad \textbf{19)}$$

When $t_i \geq t_N$, the client program should start downloading data for the pie-shaped section corresponding to the avatar's FoV from Figure 2 at point (r_i, ϕ_i) on the extended event horizon boundary.

If $t_i < t_N$, then complete caching of the avatar's environment in the new destination by the time the avatar reaches the extended event horizion is not possible, but some data may be still be useful. The client and server may also want to modify the parameters of the avatar's environment, in order to reduce the avatar's speed, or reduce rs, as described in Section III above, so that the movement time more closely matches the download time. Such changes in environmental parameters may not be possible if there are many avatars in the region, since any change will be visible to all avatars.

5.1 Microscheduling of Extended Event Horizon Update

Update of the extended event horizon can be microscheduled so that areas which the avatar is likely to move into are updated first. Figure 3 illustrates. If the avatar is moving at an angle when the avatar's event horizon intersects the extended event horizon, then the update process should proceed to update preferentially in the direction of angular movement. As shown in Figure 3 by the gray arrows, update should sweep forward along vector of travel and outward from the vector of travel, so that those objects closest to the vector of travel are the first to be downloaded to the client, while those peripheral to the vector of travel are downloaded later. The numbered checkerboard pattern in the figure indicates scheduling of objects for downloading. In the figure, four objects along the vector of travel have already been downloaded.

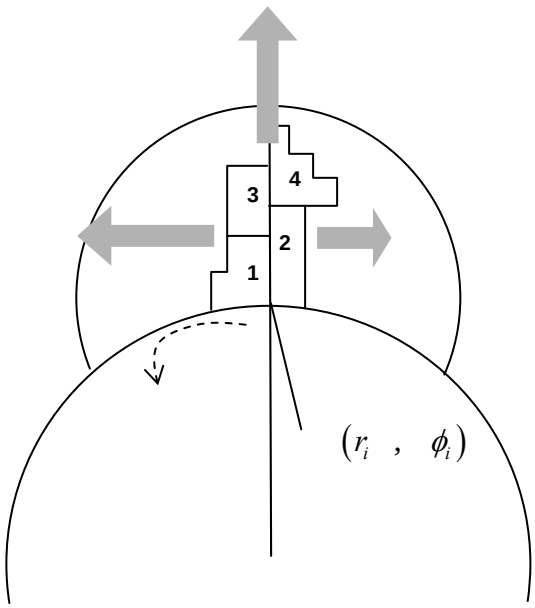

Figure 3: Microscheduling of Extended Event Horizon Update

6. PREVIOUS WORK

In general, there does not seem to be much published work specifically on optimizing graphical update for virtual reality environments, though performance considerations must certainly play a role in developing production-quality multiplayer

Digital Object Identifier: 10.4108/ICST.IMMERSCOM2007.2243
http://dx.doi.org/10.4108/ICST.IMMERSCOM2007.2243

massively parallel on-line games and shared virtual reality environments.

Ogawa, Tsukamoto, and Nishio [5] describe a system whereby the server divides the virtual space up into subspaces and repeatedly broadcasts environmental data to clients over a broadcast channel. The server waits for a short period after broadcasting one subspace update before broadcasting the next. An avatar that arrives at the boundary of a subspace just as the server completes broadcasting the latest update must wait until the update for the next subspace arrives. They describe two scheduling strategies to reduce the average waiting time for avatars moving from one subspace to another. One strategy minimizes waiting time for an avatar moving forward in the direction of the streaming update, the other minimizes waiting time regardless of when the update is expected to arrive.

Okanda and Blair [6] provide a high level review of various shared virtual reality systems. They describe a variety of methods for updating the avatar's environment, including cloning objects when an avatar gets within a certain distance and dead reckoning for calculating movement of other objects. None of the methods mentioned seem to involve statistical predictors for movement, however.

7. CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

We anticipate that a pure client side approach, while useful for generating estimates, may require changes in actual practice. For example, any modifications made to the cached environmental data on the server need to be propagated to the client prior to the rendering of the cache. While modifications from other avatars and dynamic movements such as wind moving trees, etc. are likely to be well-handled by the client-server protocol, since they can also occur within the avatar's event horizon, modifications that involve larger changes - such as someone suddenly plopping a building down while the avatar is moving towards an area - may require additional server side support.

An important area of further work is to validate the movement model and caching algorithm to see whether they hold in an actual shared virtual reality environment. We are planning to instrument the client program for Second Life [4] to test our algorithm. This should reveal how useful environment update caching is in practice.

The Kalman filter movement model presented above is entirely in Cartesian co-ordinates, with a single conversion made to polar co-ordinates for the maximum velocity constraint. A more natural model keeps the measurements in polar co-ordinates, but the resulting Kalman predictors become somewhat complicated because the covariance matrix corrector and the state variable corrector must account for coupling between the x and y co-ordinate updates due to the conversion from polar to Cartesian co-ordinates. There are a variety of ways the Kalman filter can be modified to reflect this change, which are used in radar tracking [7].

Finally, the same tracking approach using a Kalman filter that is described here could be used on the client side to track movements for objects in the environment, such as other avatars. This could reduce the amount of data traffic required for updating the local environment display for dynamically moving objects. There are limits on how effective purely physics-based tracking models such as the Kalman filter can be. For example, it would not work for tracking fine movements such as gestures, facial expressions, etc. for avatars that are nearby, but it might be sufficient for tracking other avatars that are moving toward the user's avatar from a distance, and for vehicles and other nonanimate objects that naturally obey physics movement laws.

8. REFERENCES

[1] "Kalman Filter", Wikipedia, http://en.wikipedia.org/wiki/Kalman_filter

[2] Welch, G., and Bishop, G., "An Introduction to the Kalman Filter" SIGGRAPH 2001, Course 8, http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf

[3] Ramachandra, K.V., Kalman Filtering Techniques for Radar Tracking, Marcel Dekker, Inc., 2000.

[4] http://www.secondlife.com

[5] Ogawa, T., Tsukamoto, M., and Nishio, S., "Virtual Space Broadcasting based on the Speed of Avatar Movement", Proceedings of Internet and Multimedia Systems and Applications, Grindelwald, Switzerland, 2005

[6] Okanda, P., and Blair, G, "Analysis of Techniques used in Distributed Virtual Environments," Computing Department, Lancaster University, Internal Report MPG-02-01

[7] Li, X. R., and Jilkov, V., "A Survery of Manuevering Target Tracking-Part III: Measurement Models", Proceedings of SPIE Conference on Signal and Data Processing of Small Targets, San Diego, CA, July, 2001

Digital Object Identifier: 10.4108/ICST.IMMERSCOM2007.2243
http://dx.doi.org/10.4108/ICST.IMMERSCOM2007.2243