Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

PRE-RECKONING ALGORITHM FOR DISTRIBUTED VIRTUAL ENVIRONMENTS

Thomas P. Duncan
Denis Gračanin

Department of Computer Science
Virginia Tech
7054 Haycock Road
Falls Church, VA 22043, U.S.A.

ABSTRACT

This paper proposes a pre-reckoning algorithm for distributed virtual environments. First, an overview of dead reckoning techniques used in distributed virtual environments is provided. The benefits and drawbacks of implementing dead reckoning within specific types of distributed virtual environments are discussed. An alternative to traditional dead reckoning techniques used in DIS-compliant distributed virtual environments is proposed. The alternative, referred to as "pre-reckoning," seeks to significantly reduce prediction error with a minimal increase in the number of entity state update packets issued. The performance of the pre-reckoning algorithm is compared to one of the DIS standard algorithms for location prediction. The test cases are based upon a game-based environment where the movements of participants are influenced by physical boundaries.

1 INTRODUCTION

Dead reckoning is used within distributed virtual environments to manage the dynamic shared state, see Singhal and Zyda (1999). The objective is to minimize the number of entity state update packets exchanged between participants across the network while maintaining a reasonably consistent view of each other's state. Reducing the dependency upon the network improves the scalability of the distributed virtual environment by enabling a greater number of objects to be modeled before the network becomes a bottleneck, see Cai et al (1999).

In distributed virtual environments that employ dead reckoning, the behavior of an entity is modeled by other participants referred to as *remote hosts*. The behavior modeling typically focuses upon the entity's location, speed and direction of movement, and orientation. Rather than solely basing the displayed behavior of an entity upon the update packets received across the network, the remote hosts execute a predictive model to fill in the gaps between updates. Since the display updates are based upon locally computed predictions of the actual state, dead reckoning can achieve a smoother rendering of an object's behavior. Conner and Horing (1997) refer to this as providing a "low latency experience" from the users' perspective since the update of the display does not wait for information from the network.

In a distributed virtual environment that does not use dead reckoning, the representation of an entity on a display only changes when an update is received. Limitations in network bandwidth or numerous routing hops as can be experienced across the Internet may result in sustained update rate on the order of 5 per second. Studies have shown that an update rate of 30 frames per second is required to achieve a rendering that appears smooth to the human eye. While there is a distinction between the display update rate and the entity state update rate, i.e., the display can be updated at a rate faster than updates are received across the network simply be repeating the displayed image, if an entity's behavior is only updated 5 times per second, then the displayed image may appear jittery if the discernable appearance, location, or speed of the entity is changing at a faster rate.

While dead reckoning allows a discrepancy between the actual behavior and the predicted behavior, Singhal and Zyda (1999) assert that dead reckoning sacrifices consistency in order to allow the dynamic shared state of the distributed virtual environment to change more frequently. However, while there is always a degree of error between the predicted and actual behavior, all remote hosts can achieve a consistent view of an entity's state since they are executing the same predictive algorithm. The view consistency depends upon the reliability of the network and the protocol used to convey entity state updates.

The remainder of the paper is organized as follows. Section 2 provides an overview of dead-reckoning algorithm. Section 3 describes the proposed pre-reckoning algorithm. Section 4 evaluates the proposed algorithm and compares it with the standard dead-reckoning algorithm. Section 5 concludes the paper and provides directions for future research.

2 DEAD-RECKONING ALGORITHM

While dead reckoning can achieve a degree of consistency among the predicted state maintained by remote hosts, it inherently permits inconsistency to exist between the actual state of an object and the predicted state maintained by other participants. While some degree of inconsistency may be acceptable, there are two main drawbacks:

1. A convergence algorithm must be executed to correct the inconsistency; and
2. The existence of inconsistency complicates interactions between entities such as agreement on collisions.

While it is generally more desirable to implement dead reckoning with an explicit convergence algorithm than without, the implementation of a convergence algorithm represents an additional degree of complexity. First, the algorithm consumes computational resources. Second, the algorithm must determine a reasonable period of time over which the entity's representation can be brought into alignment with the most recently reported state. Figure 1 illustrates the difference between immediate convergence and time-phased convergence. The path designated as Option 1 represents an immediate convergence where the most recent update to the location is displayed in the next frame. While this approach is simple and does not consume additional computational resources, it can result in a jittery display of the entity's movement. To achieve a smoother rendering, an alternative approach is to more gradually align the predicted location with the most recent update as illustrated by the path labeled as Option 2. However, this approach is not without its shortcomings. By taking additional time to perform the convergence, Option 2 allows the error to persist and possibly worsen.

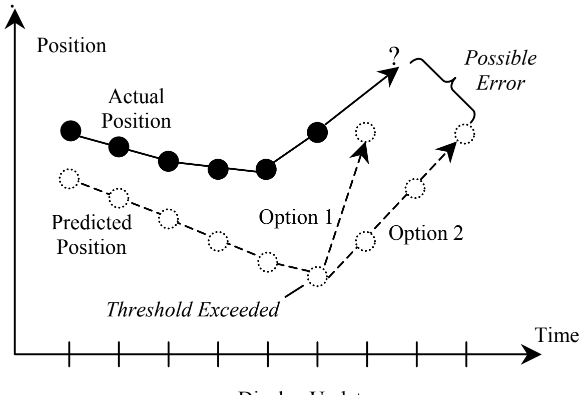

Figure 1: Convergence with Updated Location

In an effort to reduce this residual error, another alternative is for the convergence algorithm to predict the location of the entity at the completion of the convergence timeframe and then align the entity's location with that position rather than the location identified in the most recent update. Figure 2 depicts two options for convergence to the predicted location. The path labeled Option 3 reflects linear convergence similar to Option 2, while Option 4 depicts a curve fitting technique that is intended to improve the rendering of the entity's movement by eliminating any sharp turns. Under both Options 3 and 4, it may be necessary to accelerate the movement of the entity for it to "catch up" to the predicted location and then return it to the last known velocity. The convergence algorithm presented in the DIS standard does not explicitly address how entity state updates are regulated while convergence is being performed (IEEE 1995). As a result, there is a potential for entity state updates to be issued at a much higher rate than desired for an extended period of time until the predicted entity is within the error threshold.

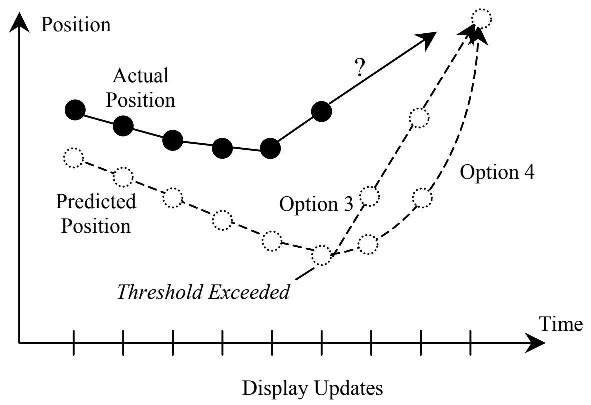

Figure 2: Convergence with Predicted Location

Another challenge associated with the convergence algorithm is that it may be asked to correct a prediction that is not easily undone. For example, Figure 3 shows a distributed virtual environment where entities move through a maze. If the dead reckoning algorithm predicts that the object will continue to move straight ahead when it in fact turns right down a corridor, then the convergence algorithm is forced into undoing the error without having the entity appear to move through a wall. While one option to limiting this type of error would be to reduce the error threshold, it would have the undesired consequence of dramatically increasing the number or entity state update packets sent across the network.

The use of dead reckoning complicates interactions between participants because there is likely to be an inconsistent view of the locations of entities in an absolute sense as well as relative to each other. This makes it difficult to agree on collisions between entities and whether weapons fired at another entity actually struck the intended target and caused the expected degree of damage. To overcome these difficulties, some applications define explicit agreement protocols or increase the update rate when objects are in closer proximity, see Cai et al. (1999). As a result, the traffic on the network can experience localized bottlenecks in areas where there is intense interaction.

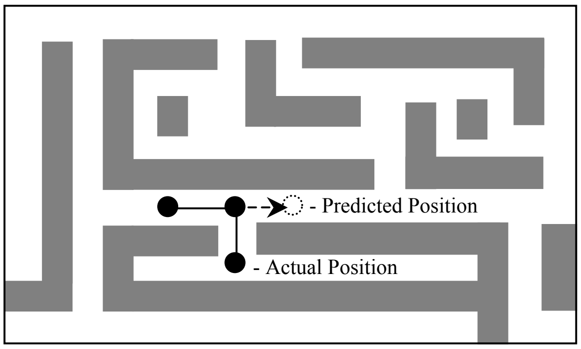

Figure 3: Difficulty in Correcting a Prediction Error

Two early distributed virtual environments, SIMNET and NPSNET, employ dead reckoning techniques that became the basis for the dead reckoning algorithms defined within the DIS standard, see IEEE (1995). The DIS algorithms are widely used for prediction of entity position based upon the reported position, velocity and acceleration. These measures are combined to form *derivative polynomials* where position is the zero order derivative, velocity is the first order derivative, and acceleration is the second order derivative, see Macedonia et al. (1995).

In addition to position dead reckoning, the DIS standard includes algorithms for predicting an entity's three-dimensional orientation based upon the angular velocity of a specified set of body coordinates. These algorithms perform what is referred to as *rotational dead reckoning*. The DIS standard defines two versions of each algorithm – one based upon the coordinate system specific to the individual entity and one based upon the coordinate system for the distributed virtual environment as a whole (world view).

For convergence, the DIS standard defines a simple linear convergence method where the number of "smoothing points" is left to the discretion of the implementer.

The DIS standard algorithms are also the basis for variations and extensions discussed in the following sections that have been developed for specific applications and operating environments.

Hybrid methods selectively employ a combination of first and second order derivative models depending upon the observed behavior of the entity. Position History-Based Dead Reckoning (PHBDR) is a widely used hybrid method where the selection of the derivative polynomial is based upon the recent history of position updates, see Singhal and Cheriton (1994). If the algorithm determines that the entity is making sudden changes in the direction of its movement, then the acceleration term is ignored and the first order derivative (velocity) is used. Otherwise, when the entity is moving in a smoother path, then the second order derivative polynomial (including velocity and acceleration) is used.

PHBDR is referred to as an *adaptive algorithm* because it adapts to match the current behavior of the entity in Cai et al. (1999). The adaptation pertains to the data used as the basis for the position prediction. The following paragraphs describe other adaptive algorithms that adjust the error threshold or the update rate depending upon conditions observed by the controlling host.

Adaptive algorithms represent the state-of-the-art with respect to *general purpose* dead reckoning algorithms. They are based upon the premise that "one size does not fit all" and there are situations where adjusting either the error threshold or update rate can improve performance (Chan 2001). Cai et al. (1999) describe an adaptive algorithm where the error threshold is reduced for objects in close proximity and increased for distant objects. The result is an overall net reduction in the number of update packets issued because there are typically fewer objects that are considered to be in close proximity. Another benefit is that it reduces the error between the actual and predicted positions of objects in close proximity which leads to improved interaction and collision detection. These adaptive algorithms are referred to as *general purpose* because they can be applied to any entity type.

Another category of dead reckoning techniques that can be considered state-of-the-art are algorithms that are tailored to the specific entity being modeled. Under this approach, different algorithms are used for aircraft than are used for land-based vehicles. The rationale for tailoring the algorithms is clear: better prediction can be achieved by making use of knowledge about the specific entity. While executing a different set of algorithms for every type of entity potentially requires additional computational resources, there are benefits as well. The modeling can be improved and possibly simplified in some ways by applying the known physical constraints of the object, e.g., the turning radius of a tank.

In addition to specialized algorithms for specific types of vehicles, there is extensive research into the dead reckoning of articulated objects – particularly humans. An articulated object is a complex entity comprised of numerous appendages (e.g, arms, legs, hands, fingers) that can function independently and have their own rotation axes, see Capin et al. (1999). The dead reckoning of human participants is particularly important to the continued development of distributed virtual environments because it directly influences user satisfaction with the virtual environment.

Two basic approaches to the dead reckoning of humans are to: (1) explicitly track and model the movement of the individual appendages; or (2) define scripts that describe a predefined set of human movements, e.g., standing up from a sitting position, and execute the script when a particular behavior is initiated. Both techniques have been shown to be effective depending upon the application. The script-based approach can achieve very fluid motion, but the error can be more extensive and difficult to undo if the wrong script is executed.

The final set of dead reckoning techniques that can be considered state-of-the-art are designed to optimize a particular metric. Two such algorithms are cost optimization-based dead reckoning and dead reckoning based upon data freshness requirements, see Holbrook et al. (1995).

The concept behind cost optimization-based dead reckoning is to minimize the *total information cost,* in Wolfson et al. (1999). Comprising the total information cost are the "costs" of prediction error, resources used in disseminating an update, and uncertainty costs. The algorithm is adaptive and can be applied to thresholds for uncertainty related to the object's speed, the position prediction error, or the uncertainty as to whether the object is still functioning within the area of interest.

In dead reckoning based upon data freshness requirements, the algorithm is tailored to the dynamic nature of the object. The rate of issuing heartbeat packets is a function of how frequently the object typically changes state. For terrain entities that change appearance infrequently, a less frequent update rate is used. The approach has been shown to reduce network traffic while continuing to support the required level of interaction.

3 PROPOSED PRE-RECKONING ALGORITHM

As a variation to the published dead reckoning techniques, an algorithm is proposed wherein the controlling host uses its copy of the predictive model to anticipate changes that will likely result in the error threshold being exceeded. When this situation is detected, the controlling host issues an entity state update immediately rather than waiting for the threshold to be exceeded. The objective is to eliminate foreseeable error with no appreciable increase in update packets. While some unnecessary updates may be issued in cases where the entity reverts to the predictable behavior (and wouldn't have exceeded the threshold), it is conjectured that particular applications where discrete behavioral choices are required (e.g., the selection of a path through a maze) may benefit from this proposed variation.

To implement the proposed algorithm, the criteria must be defined for when an immediate update should be issued. At a high level, it is described as change in movement that would ultimately result in the error threshold being exceeded for the difference between the entity's predicted location and its actual location. In more specific terms, the objective is to mathematically define the conditions under which the movement is unpredictable and an update is required.

There are three types of behaviors that are candidates for pre-reckoning: (1) when an entity at rest begins to move, (2) when a moving entity comes to a stop, and (3) when an entity makes a sharp turn. The first two behaviors are easily detected, but what constitutes a "sharp turn" requires pre-defined criteria.

In specifying the PHBDR algorithm, Singhal and Cheriton (1994) define an angle of embrace based upon the three most recent position updates. When the angle is acute, the movement of the entity is assumed to be changing sharply and so the tolerated error threshold is reduced. When the angle is obtuse, the entity is assumed to be moving in a more conventional manner and the default version of the algorithm is applied. The process for deriving the angle of embrace from the most recent position updates is depicted in Figure 4.

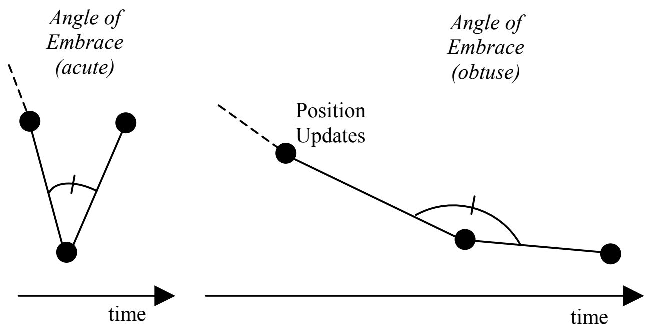

Figure 4: Position Updates for Angle of Embrace [8]

The algorithm proposed in this paper borrows upon this concept of the angle of embrace to determine when an immediate update should be issued. Since the intended application is for environments where objects may make sharp turns, the threshold for the angle of embrace is arbitrarily set at 100 degrees. The rationale for the selected value is that an angle of 90 degrees corresponds to a perfectly straight right or left-hand turn, but the entity may not always turn at perfect right angles.

The frame update rate is set to 30 frames per second and the error between the predicted position and the actual position is computed at each frame to determine if it is still within the threshold. If the error threshold is not exceeded within 5 seconds, the controlling host issues a heartbeat packet with the current location. A 100-millisecond network latency is assumed for all entity state updates. The amount of network latency influences the prediction error because of the lag between a change in the actual entity's position and when remote hosts become aware of it.

Remote hosts are expected to correct their predicted locations each time an update is received – regardless of whether the error threshold was exceeded. An accelerated convergence algorithm is employed with both the DIS standard dead reckoning algorithm and the pre-reckoning algorithm. The same convergence algorithm is used for both algorithms to ensure comparability of the results. The convergence algorithm is referred to as *accelerated* because it allows the predicted entity to move at the maximum of its previous velocity and the velocity reported in the most recent update. The predicted entity moves at this velocity until it is within an "acceptability range" of the actual entity position. For all of the experiments, an acceptability factor of 50% of the error threshold is used. The rationale for continuing accelerated convergence after the entity is within the error threshold is to prevent the threshold from quickly being exceeded again which could result in a higher than desired rate of entity state updates. The accelerated convergence algorithm is described in detail below. The actual position, trajectory, and velocity are known from the most recent entity state update that triggered the accelerated convergence.

1. Set velocity to the maximum of the previous predicted velocity and the actual velocity.
2. Calculate the distance between the previous predicted position and the actual position.
3. Calculate the time that it would take the predicted entity to reach the actual position.
4. Calculate the position of the actual entity if it were to continue moving at the same velocity and on the same trajectory. This is referred to as the *convergent position*.
5. Define a trajectory for the predicted entity to intersect with the convergent position.
6. When the predicted entity is within the acceptability range, revert to the velocity of the actual entity (if different) and follow a trajectory parallel to the actual entity's last known trajectory.

The convergent position is recalculated for subsequent frame updates if the predicted position does not converge with the actual position.

4 EVALUATION OF PROPOSED ALGORITHM

Performance of the proposed pre-reckoning algorithm is evaluated using a discrete event simulation model of the maze shown in Figure 5. The maze is based upon the layout of a Belgium mint building that has been defined for Mission 9: Operation Red Wolf of Tom Clancy's Rainbow Six computer game, see Knight (1999). Rainbow Six supports multiplayer games and the layout of the mint is representative of a virtual environment through which human-controlled commando avatars would move. The *Start* position corresponds to the front door of the mint and the *End* position corresponds to a loading dock exit in the rear. The scale of the virtual environment is roughly 100 X 100 feet.

To simplify the modeling of the environment, stairwells to other levels are treated as barriers so that movement is limited to two dimensions. In addition, the doors opening into rooms are modeled as fixed obstructions that do not move. This is a simplifying assumption that does not affect the analysis of the algorithms. The mint layout was selected because it has open spaces as well as narrow corridors that can be used to observe the performance of the algorithms under varying conditions.

For the initial set of tests, the physical boundaries are taken into account when defining actual entity movement, but they are ignored when calculating the predicted position.

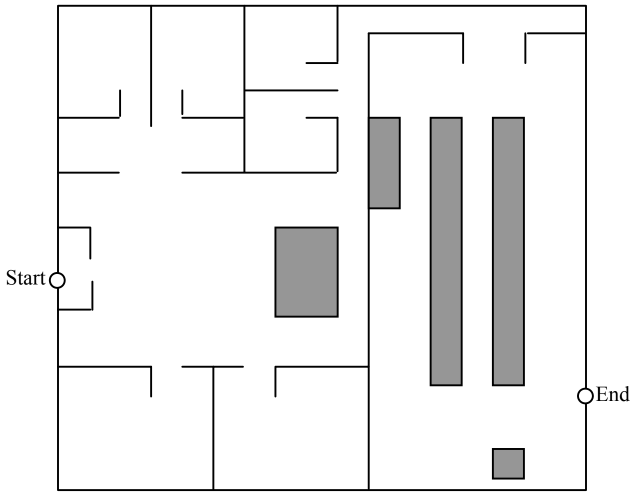

Figure 5: Environment Used for Algorithm Evaluation

While this can result in the predicted entity moving through walls, the objective in the first phase of experiments is to observe the general performance of the pre-reckoning algorithm in comparison to the DIS standard dead reckoning algorithm. Although the physical boundaries are ignored, when the predicted movement is overlaid onto the test environment, it can be observed when the inaccuracy of the predictive algorithms would result in indirect convergence, i.e., the convergent path would have to be routed around physical obstacles.

Three different "actual" routes are modeled through the maze that correspond to different degrees of variability with respect to the entity's movement. The objective is to evaluate the pre-reckoning algorithm under a range of movement behaviors.

For the first set of experiments, the actual entity moves from the start position to the end position along a direct path without stopping. Minor changes in velocity occur as the entity changes direction. The range of the velocity of the actual entity within all of the experiments is approximately 0-10 feet per second. This range was selected to reflect rates of movement that a typical human could achieve under "real world" conditions. Based upon the path defined for Experiment 1, the time for the actual entity to move through the environment is 63.5 seconds.

For the second set of experiments, a path is defined with moderate variability. The actual entity looks in rooms, but does not enter them. In some cases, the actual entity moves behind an obstacle as a defensive measure. In two instances, the actual entity comes to a stop (depicted by a solid square on the diagram of the path) and then restarts its movement.

Under the second set of experiments, the actual entity also changes its velocity without changing its trajectory. These occurrences are depicted by a solid circle on the diagram. The actual time to move along the complete path from Start to End is 75.9 seconds.

In the third set of experiments, the actual entity moves in a pattern indicative of searching the environment while having to evade hostile entities. As in the test cases with moderate variability, the actual entity changes velocity, comes to a stop, and then starts moving again. The added dimension for these test cases is that the changes in trajectory and velocity are more frequent. The zig-zag motion as the entity approaches the End position is intentionally designed to observe the behavior of the pre-reckoning algorithm when the actual entity makes a series of sudden changes in trajectory. The actual time to complete the movement along the path is 152.1 seconds.

For each route, the performance of the pre-reckoning algorithm is compared with a DIS standard first order derivative algorithm. To simplify the evaluation of the proposed algorithm, acceleration is not modeled and so changes in velocity are instantaneous. The test cases corresponding to each route are run for three different error thresholds: 0.5, 1.0, and 2.0 distance units where each unit translates to approximately 6 feet. The thresholds correspond to 0.3%, 1.1%, and 4.3% of the total space of the virtual environment, respectively. The objective is to determine if the performance of the proposed pre-reckoning algorithm is affected by the magnitude of the error threshold and, if so, what values yield the best results. The complete set of test cases is defined in Table 1.

Table 1: Description of Test Cases

Test Case	Route Variability	Threshold
1	Direct Path	0.5
		1.0
		2.0
2	Moderate Variability	0.5
		1.0
		2.0
3	Evasive Path	0.5
		1.0
		2.0

The configuration of the simulated environment is the same for each test case. Both dead reckoning algorithms are metered to produce an update of the predicted path 30 times per second. The heartbeat update is issued after 5 seconds if nothing has occurred to trigger the issuance of an entity state update packet. These values reflect the typical ratio of update frames to heartbeat packets found in distributed virtual environments that use dead reckoning.

The simulation is executed from the perspective of the controlling host. Since the actual entity movements are known by the controlling host, they are pre-loaded into the event list. The movements are defined as discrete changes in trajectory and velocity at specific points in time. At the start of the simulation, the controlling host generates an entity state update that is received by remote hosts. The controlling host accounts for the network delay and begins its execution of the predictive algorithm. It is assumed that the delay in processing entity states updates is the same for all remote hosts. For this assumption to be valid, the hosts must be relatively homogenous and they must be applying a synchronization scheme to maintain an approximation of a global clock. Maintaining a consistent view of time is a challenge faced by all distributed virtual environments – especially those that utilize dead reckoning algorithms to predict actual behavior.

The predicted position is calculated at a frequency equivalent to the display frame rate. Each of these predicted positions is referred to as a "predicted frame." Upon processing each predicted frame, the next predicted frame event is scheduled and added to the list with the actual entity movements.

For the test cases where the physical boundaries are imposed, a *bound* event is scheduled if, proceeding on the current trajectory, the predicted path would intersect a physical boundary before the time of the next frame update. In those cases, processing of *bound* event results in:

1. The predicted path stopping at the boundary until the error threshold is exceeded and convergence must be performed; or
2. If convergence is already being performed, then the predicted path will move laterally along the boundary in the direction of the point of convergence.

The performance of the proposed pre-reckoning algorithm is evaluated based upon the cumulative prediction error and the number of update packets generated. These statistics are also computed for the DIS standard algorithm for purposes of comparison. Since the cumulative error is measured at the same frequency as the frame rate, the cumulative error will generally be larger for the experiments with the longer completion times. For this reason, the comparison of the results for the different routes is based upon the average error per measurement and the average number of entity state updates per second.

Table 2 summarizes the performance of the two algorithms for the experiments where the physical boundaries are not imposed. The pre-reckoning algorithm performs best (relative to the DIS standard algorithm) under the test cases where the error threshold was 1.0 distance units – for all three movement behaviors. However, it is not possible to generalize these results because the three paths analyzed within the experiments do not provide a broad enough sampling of possible behaviors through a physically constrained virtual environment. In all cases, both algorithms achieve a relatively low entity state update rate compared to the display rate of 30 frames per second. The average predictive error per measurement is generally less than half of the error threshold, however it is somewhat higher for the DIS standard dead reckoning algorithm – particularly when the higher error threshold of 2.0 distance units is used. These results imply that the convergence algorithm

Table 2: Comparative Summary of Algorithm Performance Without Physical Boundaries Imposed

Movement	Error	**Updates/sec.** Standard/Pre-Reckoning	**Aver. Error** Standard/Pre-Reckoning
Direct Path	0.5	3.0 / 1.4	0.4 / 0.2
	1.0	2.3 / 1.1	0.5 / 0.3
	2.0	0.6 / 0.5	0.9 / 0.5
Moderate Variability	0.5	4.4 / 2.6	0.5 / 0.3
	1.0	3.2 / 0.9	0.8 / 0.4
	2.0	1.2 / 1.3	1.4 / 1.0
Evasive Action	0.5	3.3 / 2.2	0.5 / 0.3
	1.0	2.0 / 0.8	0.7 / 0.4
	2.0	1.5 / 0.5	1.4 / 0.8

may not be effective in enabling the predicted position to "catch up" to the actual entity position once the error threshold has been violated. Additional insights may be gained from peak statistics, but that data was not collected for these initial experiments.

Comparing the results for the specified paths when the physical boundaries are not imposed, it appears that the pre-reckoning algorithm's performance improves relative to the DIS standard dead reckoning algorithm as the variability of the actual path increases. This is an expected result given that the pre-reckoning algorithm is designed to address situations where the actual entity's behavior is unpredictable. However, it is an unexpected result that the reduction in predictive error can be achieved while reducing the number of entity state updates. This result is obtained because the convergence algorithm has difficulty "catching up" to the actual entity and so entity state updates continue to be generated until the error threshold is no longer violated.

Figure 6 provides a side-by-side comparison of the pre-reckoning algorithm performance for the experiments *with* and *without* the physical boundaries imposed. The results correspond to the test cases using an error threshold of 0.5 distance units which is the only value for which complete results were available. The percentage reductions in the predictive error and the number of entity state updates are measured relative to the DIS standard dead reckoning algorithm.

As an illustration (Direct Path movement without imposed boundaries), Figure 7 provides a comparison of the predicted paths yielded by the DIS standard dead reckoning algorithm (depicted by the solid line) and the pre-reckoning algorithm (depicted by the dotted line) when the error threshold is set to 2.0 distance units. While the larger error threshold allows for significant divergence, the pre-reckoning algorithm yields a predicted path that more closely follows the direct path of the actual entity.

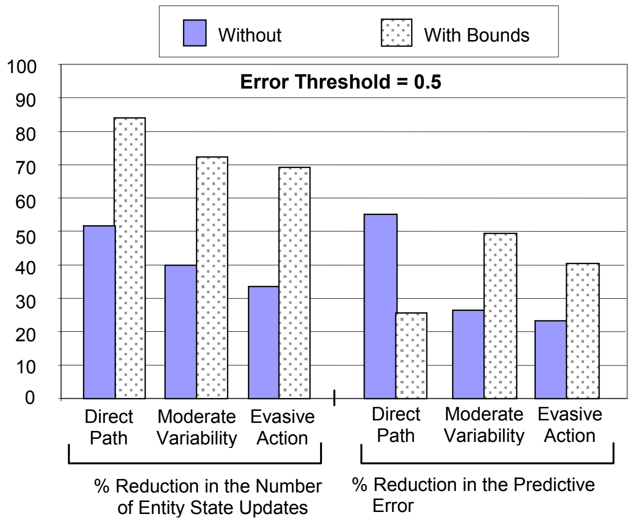

Figure 6: Summary Comparison of Pre-Reckoning Algorithm Performance

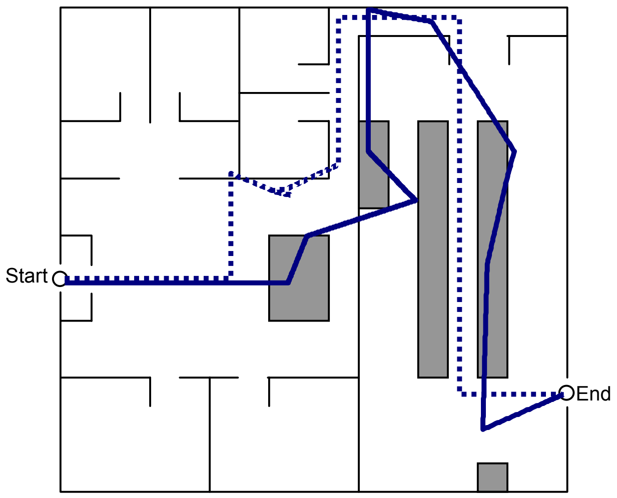

Figure 7: Direct Path with Error Threshold = 0.5

Even though boundaries are not imposed, the predicted path nearly stays within the physical boundaries. In contrast, the DIS standard dead reckoning algorithm's predicted path varies significantly from the actual path. It is important to note that the predicted entity's movement lags behind the actual entity so the actual and predicted entity are not necessarily moving at opposing trajectories at the same time. Due to the accelerated convergence algorithm, the trajectory of the predicted path is set in manner that is intended to help it converge with the actual entity path.

5 CONCLUSION

The results indicate that pre-reckoning algorithm yields a greater degree of performance improvement when there is less variability in the movement of the actual entity. In addition, the percentage reduction in entity state updates is greater than the percentage reduction in the predictive error. Both of these results are counter to the original hypothesis which expected the pre-reckoning algorithm to work better as the variability of movement increases and that the reduction in predictive error may come at the cost of additional entity state updates. Moreover, the finding with regard to variability in movement is also not consistent with the performance results when an error threshold of 1.0 distance units was used in the experiments where the physical boundaries were not imposed. As a result, a general conclusion cannot be drawn in this area because the results appear to depend upon the error threshold. The reduction in the number of entity state updates reflects the nature of the convergence that must be performed when generating the predicted path. The pre-reckoning algorithm yielded a reduced amount of predictive error in all test cases across the defined range of movement behaviors and error thresholds. Because of the "avoided" error, the pre-reckoning algorithm also reduces the need for entity state updates during the convergence process. Additional studies are required to draw broader conclusions as to the types of applications that would benefit most from the implementation of the pre-reckoning algorithm.

REFERENCES

Cai, W., F. B. S. Lee, and L. Chen. 1999. An Auto-adaptive Dead Reckoning Algorithm for Distributed Interactive Simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, ed. R. M. Fujimoto and S. J. Turner, 82-89. New York: ACM Press.

Capin, T., J. Esmerado, and D. Thalmann. 1999. A Dead Reckoning Technique for Streaming Virtual Human Animation. *IEEE Transactions on Circuits and Systems for Video Technology* 9: 411-414.

Chan, A., R. Lau, and B. Ng. 2001. A Hybrid Motion Prediction Method for Caching and Prefetching in Distributed Virtual Environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ed. M. Green, C. Shaw and W. Wang, 135-142. New York: ACM Press.

Conner, B., and L. Horing. 1997. Providing a Low Latency User Experience In a High Latency Application. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, ed. A. van Dam, 45-48. New York: ACM Press.

Holbrook, H., S. Singhal, and D. Cheriton. 1995. Log-Based Receiver-Reliable Multicast Distributed Interactive Simulation. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ed. D. Oran and S. Wecker, 328-341. New York: ACM Press.

IEEE. 1995. *IEEE Standard for Distributed Interactive Simulation – Application Protocols, Annex B: Dead Reckoning Definitions and Algorithms*, IEEE Std 1278.1-1995. New York: IEEE Press.

Knight, M. 1999. *Prima's Official Strategy Guide, Tom Clancy's Rainbow Six*. Rocklin,CA: Prima Publishing.

Macedonia, M., D. Brutzman, M. Zyda, D. Pratt, and P. Barham. 1995. NPSNET: A Multi-player 3D Virtual Environment Over the Internet. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, ed. M. Zyda, 93-ff. New York: ACM Press.

Singhal S., and D. Cheriton. 1994. *Using a Position History-Based Protocol for Distributed Object Visualization*. Technical Report STAN-CS-TR-94-1505. Department of Computer Science, Stanford University.

Singhal, S., and M. Zyda. 1999. *Networked Virtual Environments*. , New York: Addison-Wesley.

Wolfson, O., L. Jiang, A. Sistla, S. Chamberlain, N. Rishe, and M. Deng. 1999. Databases for Tracking Mobile Units in Real Time. In *Proceedings of the 7th International Conference on Database Theory*, ed. C. Beeri and P. Buneman, 169-186. Heidelberg: Springer.

AUTHOR BIOGRAPHIES

THOMAS P. DUNCAN is a graduate student at Virginia Tech. He has a B.S. degree in Applied Mathematics from Carnegie-Mellon University in 1983 and a M.E. degree in Operations Research and Industrial Engineering from Cornell University in 1984. He joined the BDM Corporation in 1984. He formed his own company, ITstrategy, in 2001 and is currently serving as a consultant to the Federal Aviation Administration's (FAA's) Telecommunication Acquisition Management Division. He is a member of ACM. His email address is <thduncan@vt.edu>.

DENIS GRAČANIN is an Assistant Professor in the Department of Computer Science at Virginia Tech. He has a B.S. and M.S. degree in Electrical Engineering from the University of Zagreb, Croatia in 1985 and 1988, respectively. He has a M.S. and Ph.D. degree in Computer Science from the University of Louisiana at Lafayette in 1992 and 1994, respectively, His research interests include virtual reality and distributed simulation. He is a senior member of IEEE and a member of AAAI, ACM, APS, SCS, and SIAM. His email address is <gracanin@vt.edu>.