How to Keep a Dead Man from Shooting

Martin Mauve
University of Mannheim, Germany
mauve@pi4.informatik.uni-mannheim.de

Abstract. The state-of-the-art approach to realize consistency in distributed virtual environments (e.g., action games, multi-user virtual reality, and battlefield simulations) is dead reckoning. A fundamental problem of dead reckoning is that participants may perceive different states for the same entity. While these inconsistencies will eventually be repaired, the resulting state of an entity may be incorrect. In this position paper we will investigate the reasons for this type of problem and propose an improved consistency approach. Our approach guarantees that all participants of a session will eventually agree on what has really happened and how the correct state of the medium should look like. In order to reach this aim we propose to transmit events as additional information, thereby allowing a simple timewarp algorithm to reconstruct the correct state of the distributed virtual environment.

1 Introduction

Both, in research and commerce, distributed virtual environments (DVEs) attracted much attention over the recent years. Examples for DVEs include networked computer games [1], multi-user virtual reality [3] and military battlefield simulations [2]. One key problem for DVEs is to maintain a consistent shared state. The current state-of-the-art approach to realize consistency in DVEs is to use dead reckoning [6]. In this paper we will demonstrate that traditional dead reckoning mechanisms may fail in ways that cause significant harm to the overall state of the DVE. Dead-reckoning does not allow to identify the situations in which it fails and, even if these situations could be identified, it does not provide the required information to repair the problem. Some DVEs may be able to accept and ignore this problem, e.g., in a large scale battlefield simulation the fate of an individual entity may not be considered decisive. However, many DVEs, such as distributed computer games, can not afford to ignore the problem.

We therefore propose to use a different mechanism to keep the state of a DVE consistent. This mechanism can be used either in combination with, or as a replacement for, dead reckoning. While it does not prevent brief periods of inconsistent state, it does allow applications to recognize and repair the problem based solely on information that is locally available. In particular no state management protocol or centralized authority is required.

2 Dead Reckoning: Why a dead man may be able to shoot

Dead reckoning is a combination of state prediction and state transmission that is commonly used to keep the state of a distributed virtual environment consistent. It is a distributed approach that does not require a centralized server. Using a distributed approach is mandatory for many DVEs in order to avoid the well known problems of centralized systems such as increased latency, single-point-of-failure and lack of scalability.

In order to use dead-reckoning the distributed virtual environment is partitioned into entities. Examples for entities are a person, a car, a plane or a bullet. Each of these entities is controlled by exactly one application that participates in the DVE. All applications that are interested in an entity are able to predict how the state of the entity will change over time. For example a plane will be predicted to change its position depending on its velocity and heading. The controller of an entity regularly checks whether the difference between the prediction and the actual state exceeds a certain threshold. If this is the case the controller of the entity transmits the state so that other applications may learn about the correct new state of the entity.

Generally there may be two reasons why the real state of an entity may differ from the predicted state. First, the prediction of remote entities may be inaccurate. This is commonly the case when the behavior of an entity cannot be fully duplicated at remote sites due to its computational complexity. This problem can be solved by providing sufficient computational resources. A much more fundamental reason for divergent states are user actions. For example, a user might have changed the heading or the velocity of an entity. This is a problem that cannot be solved by adding processing power since user actions are not predictable in a precise way. For the remainder of this work we assume that the prediction of state changes is accurate and that the only source of state divergence are user interactions.

Dead reckoning has properties that are desirable for DVEs. The first, and maybe most important one, is that it is self-healing. A lost state transmission will eventually be compensated by another state transmission for the same entity. Furthermore there is no need for a centralized management of states. Each application is able to manage the state of the entities locally by means of prediction and received state updates. Because of these properties dead reckoning is used for many DVEs.

However, there is also an important disadvantage to using dead-reckoning: at certain times applications may hold distinct states for the same entity. Because of network latency and packet loss the difference between the predicted and the real state may significantly exceed the threshold that triggers the transmission of a state for the entity by its controller. While these inconsistencies will eventually be repaired, the resulting state of an entity may be incorrect.

Let us consider two examples for this problem. Both examples are taken from a military background. The same problems can occur for distributed sport games or in multi-user virtual reality, although with a somewhat less dramatic effect.

A Dead Man that Shoots. Consider the following situation that could happen in an action game: player A shoots player B. Using dead reckoning the controlling application of player A will create a bullet entity with a certain heading and velocity. It will then transmit the state of that bullet entity. Upon receiving the state of the bullet, remote applications will start to check whether any entity that they control is hit by the bullet. Unfortunately there is a certain network delay between the time A transmits the initial state of the bullet and the time B receives the state. This network delay may be much larger (in the order of 100ms or more) than the amount of time that the bullet needs to hit its target. During this time player B might take actions, e.g., shoot at another player C, even though he should not be able to do so (because he's dead). In the presence of packet loss, the delay may increase tremendously, giving B an even longer time to take actions. Furthermore, at the time player B receives the state of the bullet he might have moved so that the bullet would not hit him. Players A, B, and C may therefore disagree about whether a hit has been scored on B or not. By simply transmitting states dead reckoning neither allows the detection of this problem nor does it provide any help in solving it. Therefore most DVEs that use dead reckoning simply ignore the problem and let the target decide whether it got hit or not.

A Flying Tank. Another example is depicted in Figure 1. In this example application A is the controller of a tank, which currently moves straight ahead. A mine is controlled by application B. The mine should explode if the tank moves over it. At some time the driver of the tank decides to change the heading to a course that would intersect the mine. As usual this will lead to a divergence between the predicted and the actual state of the tank which will cause A to transmit the new state. However for the time that is required to receive the updated state information, B will predict that the tank continues to move straight ahead. At the time the updated state is received by B, it predicts the position 1. After the new state for the tank has been examined B learns that the actual position of the tank should be 2. However, B does not know how the tank got there, so it may not trigger the explosion of the mine. As with the problem described above DVEs that rely on dead reckoning typically ignore this problem.

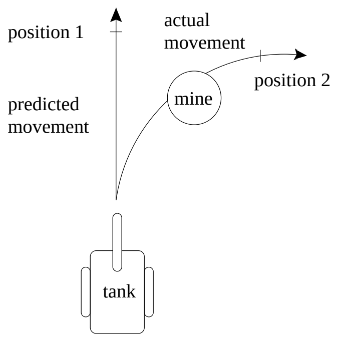

Fig. 1. Flying Tank Problem

The list of examples where dead reckoning leads to unreasonable results can be arbitrarily extended. It is therefore interesting to investigate the fundamental reason for these problems. At a first glance it seems to be the delay introduced by the network that causes the undesirable behavior of entities in DVEs. Unfortunately the delay is very hard to reduce. Steps may be taken to minimize packet loss and optimize buffering delay at routers; however, a certain amount of delay will always be present. In the Internet this amount of delay can be expected to be significant, especially in a geographically wide-spread session.

Therefore we must accept that the state of entities will become inconsistent from time to time, especially when a user controls the entity. This is not the point that we criticize about traditional dead reckoning. What we do view as a shortcoming is that traditional dead reckoning approaches undertake no efforts to detect and repair the problems that arise from these inconsistencies.

3 Timewarp: Preventing that a dead man shoots

Why is a DVE that relies on dead reckoning not able to detect and repair such problems as those described in the two examples above? The answer is simple: because applications do not have adequate information about entities that they do not control.

Consider the flying tank problem. When dead reckoning is used then the application controlling the mine simply gets the information about the tank's new position and velocity. It does not get the information when the tank turned. However without this information it cannot decide whether the tank triggered the mine or not.

In order to detect and repair problems like those described above, we therefore propose to transmit additional information from the controller of an entity to the other participants in the session. *Whenever the state of an entity changes because of external influences that cannot be predicted (such as user actions), a piece of information is produced that may be vital to the overall behavior of the DVE.* We call this piece of information an event.

Events carry a timestamp of the time they took place. This assumes that a common time base is available for all participants. This assumption is common for almost all DVEs; they typically use NTP [5] or the GPS timer. While such a common time base may not be sufficient to establish a full ordering on events (e.g., several events may have the same timestamp) this does not pose a problem. Events with the same timestamp are either interpreted to be simultaneous or an additional criterion is used (e.g., unique participant identifier) to establish a full ordering relation on events with the same timestamp.

Events are distributed to all participants of a session. There are different ways to do this which we will discuss later. For now let us assume that events will be transmitted in a way so that all participants in a DVE will eventually receive all events that are relevant to them.

With the information about events the problems described above can be avoided by using the timewarp algorithm depicted in Figure 2. It works as follows:

- Each application periodically makes snapshots of the state of all entities. Old snapshots are deleted when it becomes sufficiently unlikely that a delayed event arrives which pre-dates the recorded state (depending on the network this time will be in the order of a few seconds).
- Each application keeps a list of the events that it has received in the past. Events from this list are deleted if a state snapshot that succeeds the event is deleted.
- When a new event arrives it is inserted into the list of events.
- A timewarp is performed to the last recorded state prior to the new event. In the example this would be state 2. The current state is then re-computed based on the last recorded state and the events that have occurred since that state has been recorded, including the new event. Events that are performed after the new event may be modified by the occurrence of that event.

Let us now consider how the two examples would be handled with the timewarp approach. In the "dead man that shoots" example the event of player A firing a gun will still arrive late at player B's machine, possibly after he himself fired an illegal shot at C.

However, since all participants are guaranteed to receive both events and since they use timewarp if any one of the events arrives late, they will be able to agree on the correct state. In this case the correct state require a re-interpretation of the shoot event from participant B. All three applications will ignore this event, once they have been informed about the successful shoot event from A.

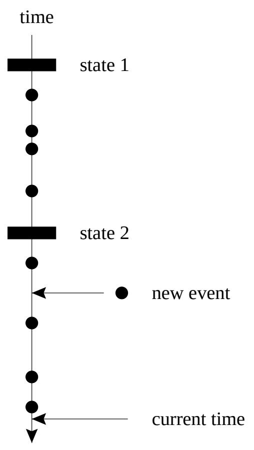

Fig. 2. Timewarp

In the “flying tank” example the controlling application of the mine will eventually receive the event that the tank has turned at a certain position. Performing the timewarp algorithm, it learns that it should have exploded and can take the appropriate actions.

The timewarp approach derives its effect from the fact that it ensures that all applications will eventually agree on what has really happened. The state of the DVE will be *consistent and correct* when all events have been delivered to all participants and all required timewarps have been executed. Even while events are being generated and transmitted the state produced by timewarp converges towards the consistent and correct state. Traditional dead reckoning approaches on the other hand converge towards a consistent state that may be incorrect. Our approach provides this service without requiring any centralized component. As long as each participant will *eventually* receive all events, only local information is required to perform the timewarp.

Two questions remain open. The first is how to guarantee that each application will really get all events. Basically this problem can be solved in two ways. The first approach is to extend dead reckoning by appending the events to the entity’s state. Whenever the entity’s state is transmitted, the list of events caused by the entity is also transmitted. Events may be deleted from this list after it becomes unlikely that a participant has not yet learned about the event. This approach can be used as an extension to the regular dead reckoning algorithm. The second approach is to use reliable transmission for events. Compared to the transmission of states this has the additional benefit that multiple participants may control (i.e., generate events for) a single entity in a joint way.

The second question concerns the complexity of the timewarp algorithm. It is clear that the timewarp algorithm has a higher computational complexity than the plain dead reckoning approach. For small numbers of entities (such as in a virtual soccer game) this does not pose an unsolvable problem. For large DVEs, however, it will become infeasible to use the timewarp algorithm for the entire DVE. As the number of entities and events grows it becomes important that the DVE is partitioned so that the timewarp can be performed with acceptable computational resources. Such a partitioning is not only required to make the timewarp less demanding but also to reduce the network traffic that each individual participant needs to receive.

4 The Road Ahead

One challenging problem that remains to be solved is how to prevent that the rendering of inconsistent state information (e.g., the erroneous explosion of a mine) distracts the user. A possible solution for this problem is to delay the rendering of critical state changes (explosion, player death, etc.) until it is likely that the rendered state change is correct. Alternatively one could try to reduce the number of situations where a timewarp is required to repair an inconsistent state.

In [4] we explored one possibility in this direction by introducing a voluntary delay before local user actions are applied to an entity. Thereby the number of short termed inconsistencies caused by the network delay can be significantly reduced. This, in turn, reduces the number of timewarps that need to be performed by the application.

In addition to preventing inconsistent information from distracting the user, a reduction of timewarps also reduces the computational effort for keeping the state consistent. The reduction of the computational complexity is the second challenge that needs to be addressed in future work. We expect that the partitioning of DVEs will play an important partin this area.

References

[1] L. Gautier, C. Diot. Design and Evaluation of MiMaze, a Multi-player Game on the Internet. In: *Proc. of IEEE ICMCS'98*, Austin, Texas, USA, 1998, pp. 233-236.

[2] IEEE Computer Society: *IEEE standard for information technology - protocols for distributed simulation applications: Entity information and interaction*. IEEE Standard 1278-1993. New York: IEEE Computer Society, 1993.

[3] M. R. Macedonia. *A network software architecture for large scale virtual environments*. Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, 1995.

[4] M. Mauve. *Consistency in Continuous Distributed Interactive Media*. Technical Report TR-9-99, Department of Computer Science, University of Mannheim, 1999.

[5] D. L. Mills. *Network Time Protocol (Version 3) specification, implementation and analysis*. *DARPA Network Working Group Report RFC-1305,* University of Delaware, 1992.

[6] S. Singhal, M. Zyda. *Networked Virtual Environments Design and Implementation,* ACM press, New York, 1999.