Local status history based interpolation algorithm for networked racing game

Niu, Junpeng

2009

Niu, J. (2009). Local status history based interpolation algorithm for networked racing game. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/19302>

<https://doi.org/10.32657/10356/19302>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**LOCAL STATUS HISTORY BASED INTERPOLATION
ALGORITHM FOR NETWORKED RACING GAME**

NIU JUNPENG

SCHOOL OF COMPUTER ENGINEERING

2009

LOCAL STATUS HISTORY BASED INTERPOLATION ALGORITHM FOR NETWORKED RACING GAME

NIU JUNPENG

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2009

Abstract

This thesis addresses the problems of un-smooth vehicle movement and inconsistent views of the virtual world in network based racing games caused by network latency and bandwidth limitation. These problems significantly affect game experience of the players.

We propose two methods to provide smooth vehicle movement without losing much consistency under most network conditions.

The first method is called Local Status History Based Interpolation Algorithm (LSHBIA). To predict and interpolate the future movement of another player's vehicle over the network, the LSHBIA uses the local status history of that vehicle in addition to the received update information from the network. The method can provide smooth vehicle movement in network based racing game, under good network conditions where the latency is less than 600ms for most of the time.

However, under bad network conditions where the latency often exceeds 600ms, LSHBIA cannot maintain the smoothness of vehicle movement well. Another method, AI enhanced LSHBIA, is then proposed. The method uses AI techniques to control the movement of other players' vehicles when the latency is greater than 600ms. Otherwise it will use the same approach in LSHBIA to handle the movement of vehicles. The method can still provide players with smooth vehicle movement in their network based racing games even when the network condition is bad.

Acknowledgements

The author would like to express his gratitude to Assistant Professor Zhou Suiping from the School of Computer Engineering, who has helped him with his support for the project.

The author would also like to thank the staffs and colleagues from TQ Global who had given him ideas and constructive comments on his work and helped him with various problems with the project as well as providing him with the resources he needed.

The last but not least, thanks to all the friends who have supported the author throughout the project.

Table of Contents

Abstract.....	i
Acknowledgements	ii
Table of Contents	iii
List of Figures.....	v
List of Tables.....	vii
Chapter 1 Introduction.....	1
1.1 Motivation	1
1.1.1 Barriers for network racing game	2
1.2 Objectives	4
1.3 Contributions	5
1.4 Outline of Thesis.....	6
Chapter 2 Literature Review	7
2.1 Some of the basic definitions.....	7
2.1.1 Source vehicle and remote vehicle.....	7
2.1.2 Time synchronization algorithm	9
2.2 Impact of network problems	10
2.3 Related Algorithms.....	14
2.3.1 Prediction algorithm	14
2.3.2 Dead-reckoning algorithm	16
2.3.3 GS-DR-LL algorithm	18
2.3.4 Adaptive dead-reckoning algorithm.....	20
2.3.5 Adaptive Δ -causality control and dead-reckoning control algorithm....	22
2.3.6 PHBDR algorithm	23
2.3.7 Interest scheme for path prediction	26
2.4 Convergence algorithm	28
2.4.1 Snap convergence algorithm	28
2.4.2 Linear convergence algorithm.....	29
2.4.3 Cubic spline convergence algorithm.....	30
2.5 Chapter Summary	31
Chapter 3 Local Status History-Based Interpolation Algorithm	33
3.1 Overview of the LSHBIA	33
3.2 Source Vehicle Updating Part.....	35
3.3 Remote Vehicle Receiving Part.....	37
3.3.1 Interpolation Algorithm	38
3.3.2 Filtering algorithm	42

3.3.3	Moving vehicle with local network Artificial Intelligence.....	46
3.4	Chapter summary.....	47
Chapter 4	AI Enhanced LSHBIA.....	48
4.1	Overview of the AI enhanced LSHBIA	48
4.2	The structure of the AI enhanced LSHBIA	49
4.3	The AI used in the AI enhanced LSHBIA	52
4.4	Chapter summary.....	56
Chapter 5	Experiments and Results	57
5.1	The Experiments	57
5.1.1	The Experiment Setup.....	57
5.1.2	The Experiment Method	58
5.2	Experiment results of LSHBIA and analysis	60
5.2.1	The smoothness performance testing results	60
5.2.2	The consistency performance testing results.....	64
5.2.3	The Advantages of the LSHBIA	66
5.2.4	The Disadvantages of the LSHBIA	66
5.3	Experiment results of AI enhanced LSHBIA and analysis	69
5.3.2	Advantages of AI enhanced LSHBIA.....	74
5.3.3	Disadvantages of AI enhanced LSHBIA	75
5.4	Chapter summary.....	75
Chapter 6	Conclusion.....	76
6.1	Summary	76
6.2	Limitations and Future work.....	77
References		78

List of Figures

Figure 2.1	The source and remote vehicles in the networked system	9
Figure 2.2	Player performance versus latency for game categories [23]	11
Figure 2.3	The relation of responsiveness and consistency under high network latency	13
Figure 2.4	Structure of the dead-reckoning algorithm.....	17
Figure 2.5	Structure of the GS-DR-LL algorithm.....	20
Figure 2.6	Difference between Interest scheme and dead-reckoning algorithms	27
Figure 2.7	Snap convergence algorithm.....	29
Figure 2.8	Linear convergence algorithm	30
Figure 2.9	Cubic spline convergence algorithm	31
Figure 3.1	The general structure of the LSHBIA.....	35
Figure 3.2	The structure of the LSHBIA algorithm's receiving part.....	37
Figure 3.3	The calculation of the current acceleration.....	39
Figure 3.4	The moving status of the vehicle in the interpolation algorithm.....	40
Figure 3.5	The structure of the interpolation process	41
Figure 3.6	The flow of the LSHBIA filtering algorithm.....	42
Figure 3.7	The filtering algorithm of the LSHBIA	44
Figure 3.8	Vehicle shifting problem.....	47
Figure 4.1	The structure of the AI enhanced LSHBIA	49
Figure 4.2	The track used in the network racing game	52
Figure 4.3	The track segments and connections of track segments	53
Figure 4.4	The state diagram of the AI in the AI enhanced LSHBIA	54
Figure 5.1	The experiment system with WANem installed	58
Figure 5.2	The user webpage interface of the WANem	58
Figure 5.3	The racing track used for testing	59
Figure 5.4	The smoothness of remote vehicle's movement with 100ms latency	61
Figure 5.5	The smoothness of remote vehicle's movement with 300ms latency.....	61
Figure 5.6	The smoothness of remote vehicle's movement with 600ms latency.....	62
Figure 5.7	The smoothness of remote vehicle under different network latency.....	63
Figure 5.8	The consistency performance of the 3 methods	65
Figure 5.9	The vehicle flying problem.....	67

Figure 5.10	The terrain causes the vehicle flying problem at large network latency ...	67
Figure 5.11	The vehicle shifting problem	68
Figure 5.12	The reason for the shifting problem when network latency is large.....	68
Figure 5.13	Results of smoothness testing for 1000ms network latency	69
Figure 5.14	Results of consistency testing for 1000ms network latency	70

List of Tables

Table 2.1	Network latency and online games [23]	12
Table 2.2	Subjective Impression of the Game under Different levels of latency [23] .	12
Table 5.1	Performance of AI enhanced LSHBIA with latency greater than 600ms	71
Table 5.2	Performance of AI enhanced LSHBIA under varied network condition	73

Chapter 1 Introduction

This chapter will look at the prime motivation of this project, its objectives and scope, my contributions and finally the outline of the thesis.

1.1 Motivation

Computer based games are a big market nowadays, and the market is expected to increase even more due to the growth number of players and advanced computer technologies. Multiplayer games, where several players play and interact with each other simultaneously, are more interesting and challenging than just playing with a computer program with artificial intelligence (AI). Since human players often play games more intelligently, more spontaneously and with more intuition, it is exciting and challenging if there is another human player on the other side to play with.

Network based racing game is one kind of multiplayer games. It provides players with the feeling of driving different kinds of vehicles to compete with other human players in different zones of the world. In order to give players nearly real experience of racing when they play the game, the networked racing vehicle's movements must be smooth, the views among the players should be consistent, and the responsiveness to the player's input actions (moving to different directions, breaking, acceleration, etc) should be fast.

1.1.1 Barriers for network based racing game

The two major obstacles to a good network based racing game are the network latency and the limited network bandwidth. These obstacles always exist in network based racing games and can not be eliminated.

The network latency, or delay, includes several components, such as queuing delay in the intermediate switches and routers, transmission delay across the physical medium, and processing delays at the endpoint hosts. Average one-way packet latencies among well connected hosts range from 45ms to 85ms, and the latency between hosts with poor connectivity can be up to hundreds of milliseconds. If the jitter or variation in latency is considered, the latency may be five times the normal case, which can be up to 1 second [1].

Due to network latency, any transmitted state update will be delayed by the network, and the players can not see a consistent view of the virtual world in a network based game.

In order to have an exact consistent view of the virtual world, the data transmission requires zero-latency network environment. It is not possible since the network latency can not be eliminated. That is because the speed of the light is constant, and the propagation delay itself contributes from 13ms to 45ms to the delay between the United States and Europe or Asia. There are researches being done to reduce the latency among endpoint hosts [2, 3], but these optimizations are mainly focused on the minor components of the end-to-end latency, such as the route queuing delay and the physical medium delay.

Network bandwidth refers to the data rate supported by a network connection or interface. It is usually expressed in terms of bits per second (bps). The network bandwidth is available for exchanging information among the network hosts.

In network based racing game, for each host, the exchanged information can be the entity's updated position, orientation, and velocity. Besides those, the exchange information can also be the entity's other attributes, like colour, player's information, etc. The bandwidth required to exchange the information is high, and which is even higher at the server side. Let us suppose the status of each vehicle is updated in each frame, and the frame rate is F/s , each vehicle status update is B bytes, which include position, orientation, velocity, etc, there are N players in one game session, the total bandwidth required at the server side for one game session is $N*(N-1) * B * F$ bytes. Suppose the frame rate is $40/s$, a status update packet has 36 bytes, and the number of players is 16, then the total bandwidth required for one game session is 2,764,800 bits, which is larger than 2 Mbits. The requirement of bandwidth will be even larger when the number of players is increased, or the frame update is more frequent.

Since the usage of network bandwidth is such heavy, we need to minimize it by reducing the updated packet size or decreasing the number of packets updated per second. Lots of algorithms have been investigated on these areas, such as the compression algorithms used to reduce the update packet size, and dead-reckoning algorithms [4-9] used to reduce the number of update packets.

Due to the network latency, two players over the network may have different experiences on the same network game session. For instance, player A observes that his vehicle collides with player B's vehicle on his screen. But at the same time, player B might not observe that the two vehicles collide with each other on his screen. This is the inconsistency problem.

Due to the network bandwidth limitation, when a player plays network based game with others, the status of other players' vehicles can not be updated each frame. This discontinuous status information of other players' vehicles received will cause unsmooth movement. The vehicles might suddenly jump from one position to another. This is the unsmoothness problem.

The inconsistency problem and unsmoothness problem affect the player's online game experience very much. Researches are required to overcome these problems.

1.2 Objectives

The objective of this research is to propose methods to overcome the problems caused by the network latency and bandwidth limitation. The requirements of the methods include:

(a) Smooth vehicle movement: The methods proposed should allow players to observe smooth movements of other players' vehicles over the network when they play network based racing game. It is the primary objective of the research. The methods should try to eliminate jitters and unrealistic movements of vehicles under most network conditions.

(b) Reasonable consistency: The smoothness and consistency of vehicle movements in network based racing game are trade-offs to each other. They can never be optimized at the same time, which is shown in session 2.2. Therefore, the methods proposed in this research will target satisfying consistency to the extent that is not affecting players' game experience, when the latency is small. For large latency, inconsistency is quite common among all other methods, therefore it is beyond the scope of this project.

1.3 Contributions

We propose LSHBIA, which uses the local status history of the vehicles of other players over the network and receives update information to interpolate and predict the vehicles' future movements. The method provides smooth vehicle movement without losing much consistency.

AI-enhanced LSHBIA is also proposed to improve the playability of the racing game when the network condition is bad with latency often exceeding 600ms. The algorithm uses AI to control the vehicles of other players over the network when the network latency is large. The AI-enhanced LSHBIA uses LSHBIA when the network latency is small. The algorithm ensures the vehicle movement is smooth and realistic under most network conditions.

The LSHBIA and AI enhanced LSHBIA algorithms are implemented and tested with a network based racing game, called Rally online racing game, which is still under development.

1.4 Outline of Thesis

The remaining chapters of the thesis are organized as follows: Chapter 2 discusses some of the previous works about studies on side-effects of network latency and solutions to the network latency. Chapter 3 describes the LSHBIA we proposed. Chapter 4 describes the AI-enhanced LSHBIA we proposed. The experiment methods and results on the two proposed algorithms are presented in Chapter 5. Lastly, the conclusions and directions for future research are given in Chapter 6.

Chapter 2 Literature Review

In this chapter, some of the existing researches related to our proposed methods and implementation are discussed. They fall under two categories: (1) the algorithms used to eliminate the network problems, (2) the evaluation of the network problems affects on the playability of the network based racing game.

Firstly, some basic knowledge related to the research is introduced. Secondly, the impact of the network delay and bandwidth limitation on the network based racing game is discussed. Thirdly, some of the researches used to solve the issues caused by the network problems are explained, the researches include prediction algorithms, dead-reckoning algorithms and PHBDR algorithm. At last, some of the convergence algorithms are investigated.

2.1 Some of the basic definitions

2.1.1 Source vehicle and remote vehicle

To allow the player interact with other players over the network and control the virtual world in network based racing game, the statuses of the other players' vehicles over the network and the virtual world's information must be accurately displayed. In the virtual world of the network based racing game, each player controls one vehicle. The vehicle's status can be generated based on the player's inputs. We call each player's vehicle on his/her own side as source vehicle (also can be local vehicle and real vehicle). The player's vehicle

displayed on the screens of other players over the network is called remote vehicle (also can be called as ghost vehicles or network vehicles). So one player's vehicle is called source vehicle when it is directly controlled by the player, and the vehicle is called remote vehicle when the vehicle is displayed on the other players' screen. The relationship between the source vehicle and remote vehicle in a network based racing game is shown in Figure 2.1.

The source vehicle accurately represents the status of the vehicle and it is directly controlled by its owner player. The time between the owner player's action and the player's own vehicle's reaction is called responsive time. The responsive time is used to adjust the responsiveness of the game, which is the speed of the game state changing according to player's inputs. The responsiveness of the game, which is adjusted by the responsive time, is one of the factors affect the playability of a racing game.

The remote vehicle is the copy of the player's vehicle displayed on the other players' screens over the network. The remote vehicle's status is totally based on the received status from the source vehicle. In other words, the remote vehicle provides less accurate version of the vehicle's status.

Figure 2.1 The source and remote vehicles in the networked system

2.1.2 Time synchronization algorithm

Time synchronization algorithms synchronize the time in the network based game for all the players. The synchronization can be done by synchronizing each pair of players, and it can also be done by synchronizing all the players' time with a specific computer. The time synchronization algorithms can be logical time synchronization [14, 15] and physical clock synchronization [16-19].

With the time synchronized, when the remote vehicle receives the status update from the source vehicle, the remote vehicle will know what specific time the update status is received. The remote vehicle can update and correct its status more accurately based on the status update from the source vehicle. Because the time of the status update of the source vehicle is known by the remote vehicle, the consistency between the statuses of the source vehicle and remote vehicle can be improved. Some researches listed in this chapter [4, 5,

10] use time synchronization algorithm to improve the performance. The stream-based time synchronization algorithm [19] is shown in algorithm 2.1

1. Client stamps current local time (T_p) and sends to server.
2. Upon receipt by server, server stamps server-time (T_s) and sends back to client.
3. Upon receipt by client, the client stamps its current time (T_c), and then the client calculates the latency time ($T_{Latency}$).
4. Then client calculate the time difference between client and server ($T_{difference}$)
5. Use T_d to update the local clock of the client.
6. Then repeat the steps 1-4 more times (usually 5-50), pausing a few seconds each time.
7. Sort the results in lowest-latency to highest-latency order.
8. Then choose the median latency from the sorted lists.
9. All samples above approximately 1 standard-deviation from the median are discarded and the remaining samples are averaged.

Algorithm 2.1 the stream-based time synchronization algorithm

The calculation of the network latency between the client and server is through $T_{Latency} = (T_c - T_p) / 2$. The time difference is calculated through the formula $T_{difference} = (T_c - T_{Latency}) - T_s = (T_c + T_p) / 2 - T_s$. The stream-based time synchronization algorithm synchronizes all the players' time with a specific computer, which is the server of the game. When a player wants to update his/her vehicle's status to other players, the time in that player's game needs to be updated first based on the server's time.

2.2 Impact of network problems

Network delay and bandwidth limitation are two problems that can not be avoided, which is shown in previous chapter. Researches [6, 20-24] have been done to evaluate the impacts of the latency and bandwidth limitation on the network based games. Here, we only focus on the impacts on the network based racing games.

In [23], the author shows the impact of latency on different kinds of network based games. The results are summarized in Figure 2.2. From the figure, we can see that the racing game is affected by the network latency much. When the network latency increases, the game performance drops tremendously.

Figure 2.2 Player performance versus latency for game categories [23]

The author also shows the latency thresholds for each kind of network based games, which is shown in Table 2.1. The latency threshold is a value to evaluate the performance of the game under latency conditions. When the latency is greater than the threshold of a specific type of game, that type of game's performance will drop tremendously. From the table, we can observe that the latency threshold of the racing game is low and the sensitivity of the racing game is high, which means the network based racing game is affected by the latency much.

Model	Perspective	Example genres	Sensitivity	Thresholds
Avatar	First person	FPS, Racing	High	100msec
Avatar	Third person	Sports, RPG	Medium	500msec
Omnipresent	Varies	RTS, Simulate	Low	1000msec

Table 2.1 Network latency and online games [23]

The network latency causes the inconsistency between players, and unsmooth remote vehicle movement. The player may feel uncomfortable when they play the network based racing game if the inconsistency problem and unsmoothness of the remote vehicle movement problem exist. In [22], the author figures out how players feel about the network based racing game under different kinds of network latency conditions. The results are shown in Table 2.2.

Delay	Impression
500ms	Not acceptable, car can not be controlled, action and reaction do not fit together
200ms	Delay is clearly observable, the car can be controlled, it is possible to adapt the own style and get used to this situation, the overall behavior is not realistic
100ms	Acceptable if no high demands with respect to realism are needed, delay can be noticed, but hardly optically be seen
50ms	Delay can hardly be noticed, the driving behavior is basically unmodified

Table 2.2 Subjective Impression of the Game under Different levels of latency [23]

The consistency and smoothness are trade-off to each other. We can not improve the performance of both at same time when the network latency is high. The relationship between the performance of the consistency and the smoothness is shown in Figure 2.3. If the network based racing game requires very high consistency between each player, the remote vehicle's movement

must be updated immediately when the source vehicle's status is received. But the remote vehicle's actual status may not be the same as the received source vehicle's update status, and the immediate change from the remote vehicle's actual status to the source vehicle's update status may affect the smoothness of the remote vehicle movement lots, which reduces the consistency. On the other side, if the network based racing game requires high smoothness of the remote vehicle, the correction from the remote vehicle's actual status to the received update status of the source vehicle should be smooth and continuous. Then the consistency between the players can not be ensured since the correction can not be done in each game frame. All in one, the consistency and smoothness can not be guaranteed at the same time.

Figure 2.3 The relation of responsiveness and consistency under high network latency

The network packet loss problem has the similar influence on the playability of the network based racing game as the network latency problem. When the packet loss occurs, the movement status of the source vehicle can not be received by the remote vehicle, if there is no any control being carried out on the movements of the remote vehicle, the remote vehicle may keep its previous

status until new status update packet from the source vehicle is received. This causes the inconsistency problem and unsmoothness problem, which is similar to the latency problem.

Bandwidth limitation is not a problem if there is only small number of players play in a network based racing game. If the network based racing game needs to support more than thousands of players to play together at the same time, due to the bandwidth limitation, the vehicle's status can not be updated frame-by-frame and the vehicle's status update packet can not be too large. Otherwise, the updates information may use up the limited bandwidth and network congestion may happen. At that time, the network latency will be larger and packet loss ratio will increase, inconsistency and unsmoothness problems will become even worse.

2.3 Related Algorithms

From previous session, we know that network latency and bandwidth limitation problems cause the inconsistency and unsmoothness in network based racing games. Some of the algorithms used to minimize the impact of latency and bandwidth limitation are introduced in this session.

2.3.1 Prediction algorithm

The prediction algorithm is to predict the future movement of the remote vehicle based on the vehicle's current and previous movement statuses. The future movement can be modeled as a straight line or a curvature. The prediction algorithm can be done on the source vehicle's side or on the remote vehicle's side. If the prediction is done on the source vehicle's side, more

information about the vehicle's movement is available for the prediction algorithm to use. For example, the vehicle's surrounding environments, and the player's input actions, etc. But with this method, the prediction algorithm does not know how much time will be delayed to send the status update packet to the receiver and whether the packet will be lost or not when the prediction is performed. It is difficult for the prediction algorithm to decide how much time ahead to predict for the vehicle's movement. If the prediction is done on the remote vehicle's side[6], although less information is available, the prediction algorithm knows the time used to transfer the update packet, and it knows whether the packet is received or not, then it is easy for the prediction algorithm to decide how much ahead to predict. The prediction at the remote host side can be done by using following formulas: $P_t = P_c + V_c \times t' + 1/2 \times a_c \times t' \times t'$, $P_t = P_c + V_c \times t'$ Where P_t is the future position, V_c is the current velocity, a_c is the current acceleration, t' is the time between current and the predicted future. The first formula uses the velocity and acceleration information to predict for the remote vehicle's future movement, which requires more network bandwidth but provides much more smooth performance. Second formula only uses the velocity to predict the remote vehicle's future movement, which provides less smooth performance but saves network bandwidth.

With the prediction algorithm, the remote vehicle's current and future statuses can be estimated. The remote vehicle movement can be interpolated by using the current and future statuses. In some cases, the current status of the remote vehicle is different from the one received from the source vehicle's current status, then convergence algorithms can be used to smoothly change the

remote vehicle from its predicted statuses to the received source vehicle's update status.

2.3.2 Dead-reckoning algorithm

Prediction algorithm provides consistency performance in network based racing game under the network condition with latency existing. But the network bandwidth required is still quite high, and the bandwidth limitation problem still exists. In order to reduce the network bandwidth requirement without losing the consistency performance too much, dead-reckoning algorithm [7, 10] is investigated.

In dead-reckoning system, the source vehicle needs to maintain the accurate movement status of its own, and it also needs to maintain a dead-reckoning model for its own. The remote vehicle needs to maintain a dead-reckoning model to estimate the future status of the remote vehicle's movement. The dead-reckoning models for the source vehicle and remote vehicle should be the same to ensure the estimated future statuses of the vehicles are consistent. The structure of the dead-reckoning algorithm is shown in Figure 2.4.

When the source vehicle changes its movement status, the dead-reckoning algorithm first compares the actual movement status with the estimated status by the dead-reckoning model. If the difference between the two statuses exceeds a predefined threshold, a status packet update needs to be generated and sent to remote vehicles.

When remote vehicles receive the update status packet from the source vehicle, the remote vehicle uses the dead-reckoning model on its own side to estimate its future movement status.

Figure 2.4 Structure of the dead-reckoning algorithm

The dead-reckoning algorithm only needs to generate a status update packet and send to the remote vehicles when the discrepancy between the source vehicle's estimated status and accurate status exceeds a predefined threshold. It reduces the number of packet updates and the frequency of network bandwidth usage. Besides this, the consistency of the vehicles can be maintained at the same time when the network latency is small. But when the network latency is large, the consistency may not be well maintained. When the remote vehicle receives the updated status from the source vehicle, the source vehicle's status might change a lot already. If the remote vehicle's dead-

reckoning model only uses the received updated status to predict for the movement of the remote vehicle, the remote vehicle's movement might be different from the source vehicle's actual movement. That is the reason why dead-reckoning algorithm can not maintain consistency performance when the latency is large.

2.3.3 GS-DR-LL algorithm

GS-DR-LL stands for Globally Synchronized Dead-Reckoning with Local Lag [5, 25]. It is designed to solve the problem faced by the dead-reckoning algorithm, which is inconsistency when the latency is large.

To illustrate the algorithm, the author divides the inconsistency in Dead-Reckoning into two sections: The before inconsistency and after inconsistency. The divided point is the time point when the remote vehicle receives the status update from the source vehicle. The inconsistency before the remote vehicle receives the update packet is referred to as before inconsistency, and the inconsistency after the remote vehicle receives the update packet is referred to as after inconsistency.

The after inconsistency is caused by the non-synchronized clocks between the hosts in the network system. It can be solved by synchronizing the clocks of all the hosts, the synchronizing process can be done by the time synchronization algorithm mentioned in the previous session. GS-DR can be used to solve the after inconsistency problem.

Before-inconsistency is caused by two reasons. The first reason is the delay of sending state updating since the update is only issued when the position

difference exceeds the pre-defined threshold. The second reason is the existing network latency. These two problems can be solved with GS-DR-LL algorithm. The work flow of the GS-DR-LL algorithm is shown in Figure 2.5. The condition part inside the figure shown as yellow color is used to compare the current time with the actual entity movement state's timestamp plus local lag time. If current time is smaller, then the entity's movement state needs to be updated, and the algorithm can go to next step.

GS-DR-LL algorithm uses a globally synchronized clock to eliminate the after inconsistency. It also delays the updating of source vehicle's status to eliminate the before inconsistency. The algorithm could be used in the network system that requires high consistency but low responsiveness to player's inputs. Although it provides high consistency between the players, the responsiveness to player's inputs is reduced since the player needs to wait for the local lag time to expire the responsiveness of their inputs.

Figure 2.5 Structure of the GS-DR-LL algorithm

2.3.4 Adaptive dead-reckoning algorithm

In the dead-reckoning algorithm introduced in previous session, the threshold value used to control the update of vehicle's movement status is a predefined constant value. But in practice, the vehicle in network based racing game may have different interests in other vehicles around it. In the network based racing game, when two vehicles' distance is small, one vehicle may be interested in the other vehicle's movement. But when the two vehicles' distance is far, the vehicles may not see each other, the vehicles may not care about the movement of each other and they have less interest in others' movements.

When two vehicles' distance is small, the threshold value should be small in order to make sure the difference between the source vehicle's status and remote vehicle's status is small, which can provide high consistency performance. When two vehicles' distance is large, the threshold value should be large in order to reduce the number of updates to save network bandwidth usage, since the two vehicles are too far away with each other to be cared by the players. So adaptive dead-reckoning algorithm [8] is investigated and used to efficiently control the updating of information by setting the threshold value adaptively.

The adaptive dead reckoning algorithm uses the Area-of-Interest (AOI), Sensitive region (SR) and the distance between the vehicles to determine the level of thresholds. Four levels of thresholds are defined to adaptively control the rate of packet update. Level 4 is largest threshold, and level 1 is the smallest threshold. If two entities' AOIs are not overlapping with each other, the level 4 threshold is used, since they are not interested in each other. The prediction of the remote entity's status does not need to be too accurate, and the update packets can be generated and sent at a low rate. Otherwise, when the two entities' AOIs are overlapping with each other, level 1 threshold needs to be used and the update packet needs to be generated and sent at a high rate, since the two entities are interested with each other.

Adaptive dead-reckoning algorithm adaptively sets the threshold value to control the ratio of update packet generating and sending. It reduces the number of packets updated and then saves the network bandwidth. At the

same time, the consistency and smoothness performances of the game are not reduced.

2.3.5 Adaptive Δ -causality control and dead-reckoning control algorithm

The network bandwidth usage and consistency are improved by using adaptive dead-reckoning algorithm and GS-DR-LL algorithm. But due to the difference in network latency and packet loss among players, the positions of racing vehicles displayed in one player's game become more largely different from those in another player's game. The causality is another problem which can not be solved.

Adaptive Δ -causality control or Adaptive dead-reckoning control are used in the network based racing game to improve the consistency and causality of remote vehicles [26, 27]. The combination of Δ -causality control and dead-reckoning control improves the performance of consistency [26, 27]. In [4] the author further improves the causality and consistency by combining the Adaptive Δ -causality control and Adaptive dead-reckoning control together.

Δ -causality control is used for preservation of causality. When one host receives an updated data media unit (MU), the host first saves the MU in its terminal buffer. When the host's time reaches the time limit, the generation time of the MU plus a Δ -seconds ($\Delta > 0$), the MU will be selected and used for prediction process. If the MU is received after the time limit, the MU will be discarded. The dead-reckoning control is shown in the Dead-reckoning algorithm part, which is in session 2.3.2.

In adaptive Δ -causality control and adaptive dead-reckoning control algorithm, the threshold values of the dead-reckoning and the Δ -causality are dynamically changed according to the network load. When the network load is high, the threshold value can be set to a larger value, then less updates are generated to reduce the network load. Otherwise, a smaller threshold value is used to provide more updating packets, which can reduce prediction errors. The flows of the adaptive dead-reckoning control and adaptive Δ -causality control are shown in Algorithm 2.2 and 2.3.

Calculate the dead-reckoning threshold (r) based on the network load
When source entity's position is updated

 If (estimated position (P_e) – Accurate position (P_a) > r)

 Send the update packet MU

 Else

 Continue the dead reckoning algorithm

Algorithm 2.2: Adaptive Causality control and dead-reckoning control's sender side

Calculate the Δ -causality control threshold value (r) based on the network load

Get the current system clock from system (T_c)

When new updated packet (MU) is received

 Get the MU generation timestamp (T_g) from the MU packet

 If ($T_c > T_g + r$)

 Discard the received MU

 Else

 Save the MU into the terminal buffer (B).

For each MU (M) in the terminal buffer (B)

 If ($T_c > M$'s timestamp + r)

 Push out the MU and use the MU for prediction

Algorithm 2.3: Adaptive Causality control and dead-reckoning control's receiver side

2.3.6 PHBDR algorithm

Dead-reckoning algorithm always uses the most recent received update packet (MU) to predict the future behavior of a moving vehicle. The packet contains

the source vehicle's current position, velocity, or acceleration information, which are used to predict the remote vehicle's future movements. The future movement of the remote vehicle is predicted based on the vehicle's velocity and acceleration information, so it is sensitive to sporadic change of the vehicle's status. Due to the network latency, the difference between the source vehicle and remote vehicle's statuses can not be eliminated. If the difference exists between the source vehicle and the remote vehicle's velocity or acceleration, it will affect the predicted future status of the remote vehicle a lot. So the predicted status may be different from the actual status received from the source vehicle, and the remote vehicle's status needs to be corrected from the predicted status to the actual status, which causes unsmooth remote vehicle movement.

Position History-Based dead reckoning algorithm [11, 12] uses the most recent several position updates received from the source vehicle to predict the remote vehicle's future movement instead of using extra derivatives. This is to avoid the problem mentioned above. It reduces the real-time dependencies between the source vehicle and remote vehicle as much as possible by only transmitting and using the less time-sensitive information (only position information). Then the remote host can process the received status information of the source vehicle independently based on the locally perceived latency.

The process of the received position updates can be the first-order algorithm or the second-order algorithm. The first-order algorithm corrects the remote vehicle's status more rapidly, and the second-order one smoothly corrects the state of the remote vehicle slowly and smoothly. First-order algorithm only

uses the most recent two position updates, and the equations for prediction are equations 2 and 3, where the d_{12} is the time difference between t_2 and t_1

Position at time t_2 is $P(t)|_{t=t_2} = P_2$

Velocity at time t_2 is $V(t)|_{t=t_2} = \frac{1}{d_{12}} P_2 - \frac{1}{d_{12}} P_1$

Second-order algorithm uses the most recent 3 positions, which are P_0 at time t_0 , P_1 at time t_1 , and P_2 at time t_2 , where $t_1 = t_0 + d_{01}$, and $t_2 = t_1 + d_{12}$. It uses the following formulas to get the remote vehicle's position, velocity, and acceleration at time t_2 :

Initial position $P(t)|_{t=t_2} = P_2$

Initial velocity $V(t)|_{t=t_2} = \frac{d_{12}}{(d_{01} + d_{12})d_{01}} P_0 - (\frac{1}{d_{01}} + \frac{1}{d_{12}}) P_1 + (\frac{1}{d_{12}} + \frac{1}{d_{01} + d_{12}}) P_2$

Initial acceleration $a(t)|_{t=t_2} = \frac{2}{d_{01}(d_{01} + d_{12})} P_0 - \frac{2}{d_{01}d_{12}} P_1 + \frac{2}{(d_{01} + d_{12})d_{12}} P_2$

With this, the future position can be estimated by using equation

$$P(t) = P(c) + V(c) \times t + \frac{1}{2} \times a(c) \times t \times t, \quad P(t) = P(c) + V(c) \times t, \text{ where } P(t) \text{ is the}$$

predicted future position, $P(c)$ is the current position, $V(c)$ is the current velocity and $a(c)$ is the current acceleration.

First order algorithm provides more accurate status but less smoothness, whereas, second order algorithm provides less accurate status but more smoothness.

2.3.7 Interest scheme for path prediction

The dead-reckoning algorithms work well when the network latency is small (less than 300ms). In dead-reckoning algorithms, there is a deviation in real time between the exported path and the placed path, which is called export error [28]. When the network latency is small, the export error is small and it does not affect the playability much. But when the network latency is large (more than 300ms), the export error becomes large accordingly. The large export error causes big change in the remote vehicle's status in a certain time, which affects the player's experience much. Interest scheme or behavioral assumption-based prediction algorithms [9, 13, 28] are used to improve the smoothness and consistency performance under bad network conditions.

The vehicle's movements can be affected by the player's habitual behaviors and its surroundings. The interest scheme algorithm uses the interaction effects among the player and vehicle to predict the movement of the vehicle in the network based racing game.

The relationship between player and its surroundings can be classified into two categories: attraction and repulsion. The "interest" and "distance" parameters determine the intensity of attraction or repulsion. The parameter "interest" is the degree of correlation between player and the nearby surroundings. The parameter "distance" is the space between the player and its surrounding objects or other players. If the "distance" between the player and its surroundings is small and their "interests" in each other are high, then the intensity of attraction is high. Otherwise, the intensity of attraction is low. When the intensity of attraction is high, the player will likely to move towards

its surrounding objects. Otherwise, it will try to leave away from its surrounding objects.

Figure 2.6 shows the difference between the dead-reckoning algorithm and the interest scheme algorithm. We can see the movements of the player are controlled by the intensity of attraction or repulsion. The interest scheme algorithm can predict the movement better than dead-reckoning algorithm under bad network conditions.

Figure 2.6 Difference between Interest scheme and dead-reckoning algorithms

The interest scheme algorithm provides better prediction of the remote vehicle's movement under bad network conditions, but it also introduces extra computation. In [9], the author also proposes the hybrid method to predict future movements of the remote vehicle. The algorithm uses normal dead-reckoning algorithm under good network conditions (network latency is less than 300ms), but under bad network conditions, interest scheme is used, since interest scheme is not needed under steady and short delay network conditions.

2.4 Convergence algorithm

After prediction algorithm or dead-reckoning algorithm is used to predict the remote vehicle's movements, the predicted position of remote vehicle may have difference with the actual position of source vehicle. In order to make sure consistent between the remote vehicle and the source vehicle, the remote vehicle needs to move from its predicted position to the received updated position from the source vehicle. The convergence algorithms can be used to deal with this process. There are three kinds of convergence algorithms that are commonly used, which are snap convergence (zero order), linear convergence (first-order) and cubic-spline convergence (higher order) [29].

2.4.1 Snap convergence algorithm

Snap convergence algorithm directly moves the remote vehicle from its predicted position to the received actual position from source vehicle. It is simple to implement and it ensures the consistency between the source vehicle and the remote vehicle. The snap convergence algorithm works well when the network latency is small. But when the latency is large, the position error is large, the directly moving may cause "vehicle jumping", and jitter may also happen, which distracts players. The snap convergence algorithm is shown in Figure 2.7

Figure 2.7 Snap convergence algorithm

2.4.2 Linear convergence algorithm

Linear convergence algorithm moves the remote vehicle from the predicted position to the updated actual position sequentially, which is shown in Figure 2.8. The algorithm linearly interpolates the position between the predicted position and the received updating position from the source vehicle, and then moves the remote vehicle from the predicted position to the received updating position step by step. The linear interpolation is done through the formula: $P_i = (t_u - t_p)(P_u - P_p)/(t_u - t_p) + P_p$. In the formula, P_i is the interpolated position, P_u is the updated position, P_p is the predicted position, t_p is the predicted time slot, t_u is the time slot when the updated packet is received, and t_i is the interpolated time slot.

By using this algorithm, the situation of “vehicle jumping” is avoided, and the vehicle’s movement is smooth and realistic when the vehicle does not change its speed and direction too often. But when the vehicle changes its speed and direction frequently, jitter might happen and the movement of vehicle might become unrealistic.

Figure 2.8 Linear convergence algorithm

2.4.3 Cubic spline convergence algorithm

Cubic spline convergence algorithm further improves the smoothness of the remote vehicle's movement based on the linear convergence algorithm. It uses the latest two predicted positions and latest two updated actual positions to generate a cubic spline curve, and then interpolate the cubic spline curve to let the remote vehicle move through the cubic spline curve step by step, which is shown in Figure 2.9.

Cubic spline convergence algorithm improves the smoothness of the remote vehicle's movement. Even when the vehicle changes its direction sharply or changes its speed quickly, the movement of the remote vehicle is still smooth. However the computational cost of the cubic spline algorithm is high.

Figure 2.9 Cubic spline convergence algorithm

In case the source vehicle's speed and direction not change too often or too sharply, the cubic spline algorithm wastes the computational cost, linear convergence algorithm can be used to provide smooth remote vehicle movement instead of using cubic spline convergence algorithm. The adaptive convergence algorithm is proposed to handle this situation. It uses the linear convergence algorithm when the source vehicle's movement does not change frequently and sharply. Otherwise, cubic spline convergence algorithm is used. Adaptive convergence algorithm ensures the movements of remote vehicle are smooth and the usage of computational resources is not increased.

2.5 Chapter Summary

In this chapter, we discussed the impact of the network latency and packet loss is high on the performance of the network based game and player's interest. Most of the researches mentioned in this chapter provide good smoothness and consistency performance. However, the performance of smoothness is still not good enough for network based racing game using under high latency

condition. At last, some convergence algorithms are mentioned in this chapter, these convergence algorithms are used to correct the remote vehicle from its current position to the received actual position.

Chapter 3 Local Status History-Based Interpolation Algorithm

In this chapter, the proposed LSHBIA will be discussed. The reason to implement this algorithm will be investigated. The algorithm's sender part and receiver part will be shown in detail. This chapter is divided into three parts: First part is the overview of the LSHBIA; Second part is the processing flow of the sender side; Third part is how the receiver side handles the packets received from the sender side.

3.1 Overview of the LSHBIA

The algorithms described in chapter 2 use the updated information from the source vehicle to predict the future movement of the remote vehicle. These algorithms improve the consistency among players and the smoothness of the remote vehicle's movement. But when the network condition is not good, like network latency is greater than 300ms, the remote vehicle's movement may not be smooth. LSHBIA algorithm is used to improve the smoothness performance. Meanwhile, it should ensure that the consistency performance does not loss too much. The LSHBIA algorithm uses the received update information from the source vehicle, and the remote vehicle's own history statuses to interpolate and predict for the remote vehicle's future movements.

LSHBIA is divided into two parts: First part is done on the source vehicle's side, which is to generate update packets for the source vehicle's movement status and send out the generated packets; Second part is done on the remote vehicle's side, which is to handle the received update packets from the source vehicle. The whole structure of the LSHBIA is shown in Figure 3.1.

The generating and sending update packets of the source vehicle's movements are quite similar to the process of dead-reckoning algorithms. It uses the distance between the player's own vehicle and the remote vehicles on his/her screen as inputs to calculate a time threshold value. The time threshold value is used to decide whether a new update packet of the source vehicle's movement needs to be generated and sent. If current time exceeds the sum of the last packet generation time and the time threshold value, then a new update packet needs to be generated and sent to the remote vehicle's side.

At the remote vehicle's side, the historical movements of the remote vehicle that was rendered are stored into a list in ascending timing order. The received updated source vehicle's movements are also stored into a list in ascending timing order. The algorithm uses the information stored in the two lists to interpolate and predict for the future movement of the remote vehicle. Filtering algorithm and network AI process follow up to ensure the smooth and realistic remote vehicle movement.

Figure 3.1 The general structure of the LSHBIA

3.2 Source Vehicle Updating Part

Source vehicle updating process is done at the source vehicle's side when the source vehicle's movement status is changed. A variable time threshold value, which is calculated through using the distance between the player's own vehicle and the remote vehicles on his/her screen as input, is used to control the updating process. The variable time threshold value is used to reduce the bandwidth usage. When vehicles are near each other, the interest of one vehicle to another is high, and the vehicles' movements should be smooth in order to provide players with realistic game experience. In order to provide smooth performance, the time threshold value needs to be small to let more update packets be generated and sent out. Otherwise, the time threshold value

can be large to reduce the number of update packets generated and sent out, which is used to reduce the bandwidth usage.

The relationship between the time threshold value, the interest of vehicles in each other and the vehicles' distance is shown as the formula:

$T_{threshold} \propto 1 / |Interest_{A \rightarrow B}| \propto |d_{A \rightarrow B}|$, where $Interest_{A \rightarrow B}$ is vehicle A's interest in vehicle B, $d_{A \rightarrow B}$ is the distance between vehicle A and vehicle B. $T_{threshold}$ is the time threshold value used to control the update process of vehicle A and decide whether the generated packet needs to be sent out to vehicle B or not.

The generated update packet of the source vehicle's movement includes the source vehicle's position, orientation, timestamp, etc. This is different from the dead-reckoning algorithms and prediction algorithm, since their generated update packet of the source vehicle's movement also includes velocity and acceleration information.

The position and orientation are used to estimate the remote vehicle's current and future velocity, and acceleration information. In order to make sure the estimation is correct and exact, the times of the source vehicle's side and remote vehicle's side need to be synchronized. The time synchronization algorithm shown in the literature part can be used to synchronize the times for them.

3.3 Remote Vehicle Receiving Part

Remote vehicle's receiving process is done at the receiver side, which includes three steps, interpolation process, filtering process and network AI moving process, which is shown in Figure 3.2.

Figure 3.2 The structure of the LSHBIA algorithm's receiving part

The historical movements of the remote vehicle that was rendered are stored into a list. The historical movement information contains the vehicle's position, orientation, velocity and that time slot. The list contains the newest 20 records in normal and the list is called by us as the remote list. The received updated source vehicle's movement, which contains the source vehicle's position, orientation, velocity and the time slot, are also stored into a list. The list is called by us as the source list, and it usually contains the newest 20 updates. The interpolation process uses the remote list and update list to interpolate and correct the remote vehicle's movement. After interpolation, filtering

algorithm is used to smoothly correct the remote vehicle from its current position to the actual position received from the updated source vehicle's movement status. Lastly, the local network AI controls the vehicle to drive from the current position to the corrected position obtained from the filtering algorithm smoothly and realistically.

3.3.1 Interpolation Algorithm

The interpolation algorithm uses the received update movement status information from the source vehicle to interpolate the actual status for the remote vehicle. The interpolated actual status is then compared with the remote vehicle's history movement status in the remote list. The difference between the actual movement status and history movement status is filtered through the algorithm mentioned in the next session, which provides smooth correction from its current status to the actual status.

When the remote vehicle receives the movement status update packet from the source vehicle, the algorithm puts it into the update list and sorts the update list in the ascending time order. The new coming packet does not need to be the newest one, since all the received packets will be saved into the list. This is different from the dead-reckoning algorithm, since the dead-reckoning algorithm only uses the newest update packet to predict for the remote vehicle's future movement. If the received update packet is out of date, it is discarded directly by the dead-reckoning algorithm.

The interpolation algorithm uses the latest three received update statuses from the source vehicle to interpolate the latest remote vehicle's movement velocity

and acceleration. The process is shown in Figure 3.3. The process to get the latest remote vehicle's velocity and acceleration is quite similar to the PHBDR algorithm. The PHBDR algorithm uses the latest three positions information to get the remote vehicle's latest velocity and acceleration. Which is shown as

$$v_n = (P_n - P_{n-1}) / (t_n - t_{n-1}) \quad v_{n-1} = (P_{n-1} - P_{n-2}) / (t_{n-1} - t_{n-2})$$

$$a_{current} = 2(v_n - v_{n-1}) / (t_n - t_{n-2}) \quad v_{current} = v_n + a \times (t_n - t_{n-1}) / 2, \quad a_{current} \text{ is the acceleration}$$

at time slot t_n , $v_{current}$ is the velocity at time slot t_n . The position, velocity and acceleration information are represented as vectors. The current acceleration calculation can be explained with the Figure 3.3. In the Figure 3.3, $a_{current} = (v_n - v_{n-1}) / (t_{V_n} - t_{V_{n-1}})$, since $(t_{V_n} - t_{V_{n-1}}) = 1/2 \times (t_n - t_{n-2})$, and then the acceleration is $a_{current} = 2(v_n - v_{n-1}) / (t_n - t_{n-2})$.

Figure 3.3 The calculation of the current acceleration

Beside the latest velocity and acceleration information, the orientation is also calculated through using the latest two pieces of orientation information received from the source vehicle. The calculation of the orientation is done through the formula: $o_{changespeed} = conjugate(o_n \times o_{n-1}) / (t_n - t_{n-1})$, which is shown in

Figure 3.4. In the formula, $o_{changespeed}$ is the speed of orientation changing. The orientation and orientation changing information are represented as Quaternion.

Figure 3.4 The moving status of the vehicle in the interpolation algorithm

The interpolated status for the remote vehicle is compared with the remote vehicle's local movement status. The comparison process is done through the following two steps. Firstly, for each two statuses in the remote list, there is a time interval between them, the algorithm finds out which time interval the received status update packet's timestamp is in. Secondly, it uses these two statuses to interpolate the vehicle's status at that particular time slot. This process is shown as the second step shown in Figure 3.4. The interpolation can be done through formula $P_{interpolate} = (1-t)P_{predicted} + tP_{updated}$, where $P_{interpolate}$ is the interpolated position, $P_{updated}$ is the received updated position, $P_{predicted}$ is the predicted position. $V_{interpolate} = (1-t)V_{predicted} + tV_{updated}$, where $V_{interpolate}$ is the

interpolated velocity, $V_{updated}$ is the received updated velocity, $V_{predicted}$ is the predicted velocity. $O_{interpolate} = (1-t)O_{predicted} + tO_{updated}$, where $O_{interpolate}$ is the interpolated orientation, $O_{updated}$ is the received updated orientation, $O_{predicted}$ is the predicted orientation. The t in the three formulas means the weight for interpolation.

The difference between the remote vehicle's actual status received from the source vehicle and the remote vehicle's local movement status at a particular time slot is calculated and used for next stage, which is filtering algorithm. The whole interpolation process is shown in Figure 3.4.

Figure 3.5 The structure of the interpolation process

3.3.2 Filtering algorithm

Filtering algorithm uses the difference between the received updated status and remote vehicle's movement status calculated from previous step to smoothly change the remote vehicle's movement from its current status to the actual updated status. The flow of the filtering algorithm is shown in Figure 3.5. The filtering algorithm's performance is controlled by the gain values, which include the position gain and the orientation gain. The filtering algorithm's gains are set and used to control the correction of remote vehicle's movement. The gain can be varied based on the requirements of the game. If the gain is high, the correction from vehicle's current status to updated actual status is faster but not smoother. Otherwise, it is smoother but not faster.

Figure 3.6 The flow of the LSHBIA filtering algorithm

In the filtering process, four predefined values are used to control the filtering algorithm's gain values. The four predefined values are minimal error,

maximal error, minimal gain and maximal gain. The minimal error is the minimal change of the vehicle's movement status, while the maximal error is the maximal change of the vehicle's movement status. The minimal gain is the minimal percentage change of the vehicle's movement status, and maximal change of the vehicle's movement status is controlled by the maximal gain.

The filtering algorithm first gets the difference between the received update movement status from the source vehicle and the remote vehicle's current movement status. Then the algorithm uses the following formulas to get the gain value. $t = (E_c - E_{\min}) / (E_{\max} - E_{\min})$, $G = G_{\min} + t(G_{\max} - G_{\min})$. The gain value is then filtered to get the final gain result, which is $G_r = \min(\max(G, G_{\min}), G_{\max})$. In these formulas, E_c is the current error, $E_{\max}$ is the maximal error, $E_{\min}$ is the minimal error, $G_{\max}$ is the maximal gain, $G_{\min}$ is the minimal gain, and G_r is the resulted gain.

After the algorithm obtains the gain, it uses the gain to compute the corrected position, orientation, velocity and acceleration. The corrected position is calculated through formula: $P_{corrected} = P_{current} + G_p P_{error}$, where $P_{corrected}$ is the corrected position, $P_{current}$ is the current position, P_{error} is the difference between the actual position received from the source vehicle and the current position predicted. G_p is the gain to correct the position in one frame of game. The corrected velocity is calculated through formula: $V_{corrected} = V_{current} + G_v V_{error}$, where $V_{corrected}$ is the corrected velocity, $V_{current}$ is the current velocity, V_{error} and is the difference between the actual velocity received from the source vehicle and the

current velocity predicted. G_v is the gain to correct the velocity in one frame of game. The corrected orientation is calculated through formula: $O_{corrected} = O_{current} + G_o O_{error}$, where $O_{corrected}$ is the corrected orientation, $O_{current}$ is the current orientation, O_{error} and is the difference between the actual orientation received from the source vehicle and the current orientation predicted. G_o is the gain to correct the orientation in one frame of game. The corrected acceleration is calculated through formula: $a_{corrected} = a_{current} + G_a a_{error}$, where $a_{corrected}$ is the corrected acceleration, $a_{current}$ is the current acceleration, a_{error} and is the difference between the actual acceleration received from the source vehicle and the current acceleration predicted. G_a is the gain to correct the acceleration in one frame of game. The corrected vehicle movement status is used to update the remote vehicle's movement. The correction is also done on all the movement statuses stored in the remote list, whose time slot is larger than the received movement status update packet's time slot.

Figure 3.7 The filtering algorithm of the LSHBIA

The filtering algorithm changes all the status records in the remote list when the changing value of the remote vehicle's movement is obtained from the previous step. The process of changing the statuses in the remote list can be explained with Figure 3.6.

We suppose the network latency between the source vehicle side and the remote vehicle side is Δ , which is the time interval between t_1 and t_2 , t_3 and t_4 , t_5 and t_6 , and t_7 and t_8 shown in the Figure 3.7. The update rate is f , which is the time interval between t_2 and t_3 , t_4 and t_5 , t_6 and t_7 . The red line in the Figure 3.7 represents the source vehicle's movement path. The blue line in the figure represents the remote vehicle's movement path. The purple line in the figure represents the remote vehicle's movement direction before correction.

Firstly, at time t_1 , the source vehicle updates its movement status to the remote vehicle. At time t_2 , the remote vehicle receives the updated status from the source vehicle, and then it compares its status at time t_1 with the received source vehicle's status to obtain the position difference between them at time t_1 . The position difference obtained is used for correcting the remote vehicle's current movement, and updating for the remote list stored at the remote vehicle side. The remote vehicle uses the difference obtained to estimate the remote vehicle's movement correction value by using methods mentioned in this session. Then the remote vehicle uses the movement correction value to adjust its moving path, which is shown as the blue line between time t_2 and t_4 . During t_2 and t_4 , the remote vehicle's moving path is adjusted to the blue line from the purple line. The correction process will repeat if new update packet is

received from the source vehicle. From the Figure 3.8, we can observe that the moving path of remote vehicle gradually approaches to the moving path of the source vehicle as time goes on.

3.3.3 Moving vehicle with local network Artificial Intelligence

After the vehicle's movement status is interpolated and filtered through the algorithms described above, the vehicle's position and orientation changing is smooth. The local network AI is used to drive the vehicle from its current position to the corrected position obtained from the filtering algorithm. The local network AI uses the remote vehicle's velocity, orientation information as inputs to control the remote vehicle's movements. The heading direction of the remote vehicle is also considered by the local network AI to give the vehicle correct moving direction.

In some situations, the remote vehicle's heading direction is different from the remote vehicle's moving direction. As shown in Figure 3.8, the vehicle's heading direction is described as the blue line, and the vehicle's moving direction is described as the green line. Since the prediction and interpolation algorithms do not care about the remote vehicle's heading direction, the remote vehicle's movement is unrealistic and players may feel uncomfortable of the game. We call this problem as shifting problem. The shifting problem can be solved with network AI, which controls the remote vehicle's movement with the heading direction, orientation and velocity information as inputs.

Figure 3.8 Vehicle shifting problem

3.4 Chapter summary

This chapter explained the LSHBIA in detail. The structure of the LSHBIA has two parts: The process of the sender side and the process of the receiver side. The process of the receiver side is divided into three steps: interpolation process, filtering process, local network AI controlling vehicle process. Interpolation process interpolates the received statuses in the packets from the source vehicle. It uses the interpolated status to compare with the remote vehicle's history movement status to obtain the status difference. The filtering algorithm corrects the remote vehicle's current movement status with the status difference obtained from the interpolation process smoothly. Local network AI controls the remote vehicle to move from its current position to the corrected position obtained from the filtering process smoothly and realistically.

Chapter 4 AI Enhanced LSHBIA

This chapter describes the AI enhanced LSHBIA algorithm. The reason of the AI enhanced LSHBIA algorithm is discussed, the structure of the AI enhanced LSHBIA is described. At last, the AI used in the AI enhanced LSHBIA is introduced briefly.

4.1 Overview of the AI enhanced LSHBIA

The LSHBIA algorithm described in previous chapter works well when the network latency is smaller than certain amount value (we specify this value as latency threshold value (**LTV**)). The LTA is a value specified based on different type of network based racing game to determine when the AI technique needs to be used to control the remote vehicle's movement. But when the network latency exceeds the LTV, the consistency and smoothness performance of the algorithm is reduced. In order to improve the game's performance under bad network conditions, AI is introduced into the algorithm.

When the network latency exceeds the LTV, instead of using LSHBIA algorithm to control the movement of the remote vehicle, AI technique is used to control the remote vehicle's movement to make sure the remote vehicle's movement is smooth. Under this situation, the updated status from the source vehicle is still received and stored into the remote list. When unexpected situations happen, such as collision, the remote vehicle can still use the history list to interpolate and predict its current movement information and corrects

its movement accordingly. With this, reasonable consistency can still be maintained.

When the network latency is less than LTV, the AI is disabled and LSHBIA can be used again to ensure the consistency between the source vehicle and remote vehicle. AI enhanced LSHBIA ensures the remote vehicle's movement is realistic and smooth even under bad network conditions.

4.2 The structure of the AI enhanced LSHBIA

The structure of the AI enhanced LSHBIA is divided into two parts. First, the player's skill is adjusted to choose the suitable AI for it. Second, when player plays the game, the AI is used to control the remote vehicle's movement when the network latency exceeds the LTV. Otherwise, the LSHBIA is used to control the vehicle's movement instead. The overall structure of the algorithm is shown in Figure 4.1.

Figure 4.1 The structure of the AI enhanced LSHBIA

There are three AI types can be chosen to represents the player's skills, which are beginner AI, normal AI and professional AI. The beginner AI can not control the vehicle well and the time used to finish one lap is long. The normal AI can control the vehicle well and the time used to finish one lap is short and the vehicle's speed is not very high. The professional AI controls the vehicle professionally, the speed of the vehicle is very fast and the time used to finish one lap is even shorter. When player plays the racing game, the time to finish one lap is recorded and used to adjust the AI level for that player. The formula

used to calculate the AI level is $Factor_{AI} = \sum \frac{T_{player}}{T_{setting}} / n_{lap}$. In the formula, T_{player} is

the time the player finishes one lap, $T_{setting}$ is the time to finish one lap set by the system, n_{lap} is the number of laps the player has finished. The

$Factor_{AI}$ determines the AI level: When the factor is greater than 1.2, the AI level is beginner; When it is less than 0.8, the AI level is professional; And when it is between 0.8 and 1.2, then the AI level is normal. The value 0.8 and 1.2 can be changed according to the requirement of different games.

After the player gets the AI level to represent the player, the algorithm calculates the network latency by using the timestamp contained in the received update status packet from the source vehicle. The algorithm can then use the specific AI to control the vehicle when the network latency exceeds the LTV. Otherwise, the AI is disabled and LSHBIA algorithm is used to control the vehicle. These steps can be combined as Algorithm 4.1.

```

Get network latency  $t_{latency} = t_{current} - t_{packet}$ 
If (  $t_{latency} < t_{threshold}$  ) //where  $t_{threshold} = LTV$ 
    Disable the AI
    Run LSHBIA algorithm
Else
    Enable the AI and the AI will be run at least 1s

```

Algorithm 4.1: The AI enhanced LSHBIA algorithm

Through experiment, we found out that when the network latency is less than 300ms, the source vehicle and remote vehicle's movements are consistent and the remote vehicle's movement is smooth and realistic. When the network latency is between 300ms and 600ms, the remote vehicle's movement is not quite smooth, but still realistic, and the consistency is not good as well. When the latency is larger than 600ms, the remote vehicle's movement is unrealistic and the consistency is bad as well. With the experiment results, 600ms is chosen as the threshold of the network latency, which is the LTA value. If the latency is larger than 600ms, the vehicle's AI is used to control the movement of the remote vehicle instead of LSHBIA.

After the AI is enabled, the vehicle should be controlled by AI for at least 1 second. This is because when the network latency is greater than 600ms, it means the network condition is not good at that moment, and the bad network condition is likely to be happened in the next seconds. If we directly disable the AI and use LSHBIA, when the latency is greater than 600ms again after few network packet updates, the AI needs to be enabled again. In this case, the movements of the remote vehicle will be strange and jitter may happen often since the frequently exchanging between AI controlled vehicle's movement and LSHBIA controlled vehicle's movement.

4.3 The AI used in the AI enhanced LSHBIA

The AI in the AI enhanced LSHBIA used to control the vehicle's movement is investigated in [30, 31]. The race track used in the game is shown in Figure 4.2, which is divided into several segments. Each segment is connected with its previous segment and next segment, which is shown in Figure 4.3. Each segment has one or more waypoints that the vehicle must pass through to prevent short-cut situations from happening. Each segment contains the vehicle's targeted speed to control the vehicle's speed at that specific segment. Each segment also has the branch probability information when there are branches existed in the segment. The branch probability at each branch point is used to let the vehicle choose which branch to move when the vehicle reaches the branch point.

Figure 4.2 The track used in the network based racing game

Figure 4.3 The track segments and connections of track segments

With the settings mentioned above, the Finite State Machine (FSM) is used for the AI to control the movement of the remote vehicle. There are 6 states during the network based racing, which are starting state, racing state, collision state, resetting state, off-track state and airborne state. The relationships between them are shown in Figure 4.4.

Figure 4.4 The state diagram of the AI in the AI enhanced LSHBIA

In the Figure 4.4, the relationships between the states are labeled as numbers.

1. At beginning, the state is STATE_START. When the race starts, the state changes from STATE_START to STATE_RACING. In the STATE_RACING state, the AI controls the vehicle to move from its current segment to the next segment, the control mechanism is referred to [31].
2. If the AI finds obstacles in front of the vehicle, the AI will try to avoid the collision by using collision avoidance algorithm [32]. If the collision is not avoided, the state is changed from STATE_RACING to STATE_COLLISION.
3. After collision happens, if the AI can recover the vehicle and move again, the state is changed back to STATE_RACING.
4. Otherwise, if the vehicle can not move within 5 seconds, the state is changed to STATE_RESET.
5. If the track has jumps that cause the vehicle to become airborne, the state is changed to STATE_AIRBORNE.
6. The race state changes back to STATE_RACE when the vehicle lands.
7. If the

vehicle moves out of track, the state is changed to STATE_OFFTRACK. 8. If the vehicle can move back to the centerline of the track, then the state changes back to STATE_RACE. 9. Otherwise, if the vehicle can not move back in 5 seconds, the state changes to STATE_RESET. 10. If the state is STATE_RESET, the vehicle is reset to the previous waypoint shown in Figure 4.3 and the vehicle can move again, and then the state is changed back to STATE_RACE.

When the racing state is STATE_RACING, the AI navigation algorithm is used to navigate the vehicle's movement. The navigation algorithm uses each segment's predefined parameters to control the vehicle moving from one segment to the next. The segment's predefined parameters include branch probability, the segment's track width, the segment's position, velocity and acceleration range at the segment's starting point and ending point. These parameters can be obtained by training the AI or just be set by game designers.

When the remote vehicle is controlled by AI technique, the AI navigation algorithm finds out which segment the vehicle is in. After that the algorithm can find out the next segment and obtain all the parameters of both segments. The navigation algorithm uses the branch probability to determine which path the vehicle is going to move along. Third, the algorithm interpolates the moving position and velocity for each game frame with parameters of the segment. The interpolated position should be on the track. If the remote vehicle moves out of the track, the racing state is changed to STATE_OUTTRACK, and the navigation algorithm waits until the remote vehicle being reset back to the center of the track. With the AI navigation algorithm, the AI can control the vehicle moving on the track smoothly.

4.4 Chapter summary

In this chapter, the AI enhanced LSHBIA algorithm is discussed in detail. The algorithm solves the problem of remote vehicle's unrealistic and unsmooth movements when the network latency is too large. The algorithm uses AI techniques to control the remote vehicle when the network latency is larger than LTV the game specified. When the network latency is less than LTA the game specified, the algorithm uses LSHBIA to control the remote vehicle. This chapter also briefly mentioned the logic and structure of the AI used for the algorithm.

Chapter 5 Experiments and Results

In this chapter, the experiment environment and methods are described. The results of the experiments for the LSHBIA and the AI enhanced LSHBIA are recorded and analyzed. The results are also compared with the other algorithms' results. The advantages and disadvantages of the LSHBIA and AI enhanced LSHBIA are discussed.

5.1 The Experiments

5.1.1 The Experiment Setup

The algorithm is implemented and tested in a Rally online racing game, which is still in development. Experiments are done under Local Area Network (LAN) environment with a network simulation tool to simulate the different kinds of network conditions.

The network simulation tool used in the experiment is Wide Area Network Emulator (WANem) [33]. It is Linux software performing as a route in the LAN to simulate WAN conditions. The experiment system can be explained with Figure 5.1. If computer 1 wants to send a network packet to computer 2 or 3, the packet needs to go through the route (computer 4) with WANem installed first. The route is used to set the network conditions. The settings are done through the WANem's webpage interface, which is shown in Figure 5.2.

Figure 5.1 The experiment system with WANem installed

[About](#)
[WANalyzer](#)
[Basic Mode](#)
[Advanced Mode](#)
[Save/Restore](#)
[Help](#)

Interface: eth0		Packet Limit 1000 (Default=1000)		Symmetrical Network: Yes	
Bandwidth	Choose BW	Other		Other: Specify BW(Kbps) 0	
Delay		Loss		Duplication	
Delay time(ms)	0	Loss(%)	0	Duplication(%)	0
Jitter(ms)	0	Correlation(%)	0	Correlation(%)	0
Correlation(%)	0			Correlation(%)	0
Distribution	-N/A-			Gap(packets)	0
Idle timer Disconnect		Type	none	Idle Timer	
Random Disconnect		Type	none	MTTF Low	
Random connection Disconnect		Type	none	MTTF High	
				MTTR Low	
				MTTR High	
IP source address	any	IP source subnet		IP dest address	any
				IP dest subnet	
				Application port if any	any

☐ Display commands only, do not execute them

Figure 5.2 The user webpage interface of the WANem

The configuration of the computers used in experiments are Pentium D 3.2GHz, 2GB memory and with Windows XP and DirectX 9 installed.

5.1.2 The Experiment Method

The experiment is done through running the game on two computers, with a third computer as a route with WANem running. Each game host plays both roles of sender and receiver of vehicle movement status. We set the bandwidth

of the simulated network to be 1k-byte. This is to simulate the worst case of a 56k-bits bandwidth condition for 8 players in a game session.

The game is tested under different kinds of network conditions simulated by the network simulation tool. We set the latency to specific values, which ranges from 0 to 1000ms with 100ms increment, in different experiments.

The track used for the experiment is shown in Figure 5.3, which combines a few turns and straight paths. We set every experiment for one lap, and measure the vehicle movement status for every 100ms after the game starts.

Figure 5.3 The racing track used for testing

We record and compare the movement statuses of the source vehicle and remote vehicle under different network conditions. Based on the results, we analyze the consistency as well as the smoothness performance of the remote vehicles. The results are also compared with other algorithms such as dead-reckoning and PHBDR algorithm. We set an experienced player to control the

vehicle for all the experiments, which is to ensure all the experiments have nearly same testing conditions.

In order to easily compare the smoothness of the remote vehicle and the consistency between the source vehicle and remote vehicle, the game provides a feature of viewing other vehicles' movement when player plays the game. In this way, the same vehicle can be viewed on the screens of two computers reflecting the consistency performance.

5.2 Experiment results of LSHBIA and analysis

With the testing method mentioned in previous sessions, the movement statuses of source vehicle and its remote vehicle are collected. We check the smoothness performance of the algorithm by analyzing the movement status of remote vehicle. The consistency performance of the algorithm is analyzed with the movement statuses of both a source vehicle and its remote vehicle.

5.2.1 The smoothness performance testing results

We measure the smoothness of a remote vehicle's movement in the following way: for every status recorded except the first one, the difference between positions of the status and its previous status is calculated. The calculation is done through the formula: $P_{change} = \sqrt{(P_1.x - P_2.x)^2 + (P_1.y - P_2.y)^2 + (P_1.z - P_2.z)^2}$, where P_1 and P_2 are the position and previous position of the remote vehicle's status.

We perform the smoothness testing with different simulated network latencies for every 100ms time step from 0-1000ms. The results under the 100ms,

300ms and 600ms network latency are shown in Figure 5.4-5.6. The smooth performance is analyzed with the fluctuation of the testing results. If the result wave is changed sharply often, it means the smoothness performance is bad. If the wave is smooth and its change is not too sharp, it means the smooth performance is satisfied.

Figure 5.4 The smoothness of remote vehicle's movement with 100ms latency

Figure 5.5 The smoothness of remote vehicle's movement with 300ms latency

Figure 5.6 The smoothness of remote vehicle's movement with 600ms latency

In Figure 5.4, the smoothness of remote vehicles' movements of three different algorithms is almost the same. It means the smoothness performances of the three algorithms are satisfied under the low latency network conditions.

In Figure 5.5, the smoothness of the movement under 300ms network latency is shown. From which, we can see that the smoothness of the remote vehicle's movement using LSHBIA method is better than other two methods.

In Figure 5.6, the remote vehicles' movements for all the three methods are not smooth, which is because of the large network latency. But from the figure, we can still see that the LSHBIA is better than the other two methods for the smoothness performance.

Figure 5.7 summarized the smoothness of the remote vehicles' movements for the three methods under different network latencies. The difference value in the Figure 5.7 is calculated through the following formulas:

$$P_{change}(i) = \sqrt{(P(i).x - P(i-1).x)^2 + (P(i).y - P(i-1).y)^2 + (P(i).z - P(i-1).z)^2}$$

$$Smoothness = \sqrt{\left(\sum_{i=1}^n (P_{change}(i) - P_{change}(i-1))^2\right) / (n-1)}, \text{ where } P(i) \text{ is the position of}$$

remote vehicle recorded every 100ms. $P_{change}(i)$ and $P_{change}(i-1)$ is the difference between its current position and previous position. Smoothness is calculated by using all the difference values obtained during one lap of game play, which is around 1 to 2 minutes. The smaller the smoothness value is, the better the algorithm performs.

Figure 5.7 The smoothness of remote vehicle under different network latency

From Figure 5.7, we can see that the LSHBIA, dead-reckoning and PHBDR algorithm has similar smoothness performance when the latency is less than 300ms. When the latency is larger than 300ms, the smoothness performance of LSHBIA is better than the other two algorithms. We can also see that the smoothness performance gets worse with the increase of latency starting from 300ms. When the network latency is greater than 600ms, the smoothness performance of all algorithms drops significantly. As a result, all these algorithms can no more provide smooth remote of vehicle movement.

The results show that LSHBIA algorithm has better smoothness performance compared with other two algorithms. However, with network latency greater than 600ms, the smoothness of the remote vehicle's movement cannot be maintained in LSHBIA algorithm.

5.2.2 The consistency performance testing results

Besides the smoothness performances of the algorithms, the consistency performances of all the algorithms are also tested with the position information recorded. For each time slot of the recorded status, the source vehicle and remote vehicle' positions on that time slot are interpolated and compared to get the difference on that time slot. These calculated differences are used to get the overall difference, which is used to show the consistency performance. The calculation of the overall difference is shown as algorithm 5.1.

```

Get the source vehicle movement data ( $L_{source}$ )
Get the remote vehicle movement data ( $L_{remote}$ )
For each element  $e_{source}$  in the  $L_{source}$ 
    Get the timestamp  $t_{source}$  and position  $P_{source}$  from  $e_{source}$ 
    For each neighbored elements  $e_{remote}$  and  $e_{remote1}$  in the  $L_{remote}$ 
        Get the timestamp  $t_{remote}$  and  $t_{remote1}$  from  $e_{remote}$  and  $e_{remote1}$ 
        If ( $t_{remote} < t_{source} < t_{remote1}$ )
            Interpolate the  $P_{remote}$  at timestamp  $t_{source}$ 
        Else
            Continue
    End for
    Get the difference  $d_i = |P_{source} - P_{remote}|$  at timestamp of  $e_{remote}$ 
End for
Get the overall difference by using standard division method

```

Algorithm 5.1: Method to get the difference value

Figure 5.8 shows the testing result of consistency between movements of the source vehicle and the remote vehicle under different network latencies. The consistency performances of the three different algorithms, which are LSHBIA, dead-reckoning and PHBDR, are compared in the experiment.

Figure 5.8 The consistency performance of the 3 methods

From the Figure 5.8, we can see that the consistency performances of all the algorithms become worse and worse when the network latency increases. When the latency is less than 200ms, the consistency performances of all the three algorithms are similar to each other. Starting from 200ms onwards, the consistency performance of LSHBIA and PHBDR algorithms are similar, while the performance of dead-reckoning algorithm is a little better than the LSHBIA and PHBDR. When the latency exceeds 800ms, the consistency performance of PHBDR drops significantly. LSHBIA has much better consistency performance than PHBDR, while dead-reckoning algorithm performs the best among these 3 algorithms.

The performances of the three algorithms can be explained with the following: LSHBIA and PHBDR algorithms use history status to predict the future movement position. This can result in the updates of remote vehicle's status not being on time but smooth performance. The dead-reckoning algorithm directly transfers the velocity information to ensure the vehicle updates its status on time, but causes non-smoothness performance.

The results show that LSHBIA algorithm has an average consistency performance among all algorithms compared. It provides a satisfying consistency performance when the latency is smaller than 600ms.

5.2.3 The Advantages of the LSHBIA

LSHBIA provides smoother remote vehicle's movement for network latency below 600ms. With the improvement on smoothness, the algorithm keeps a middle level of consistency performance among difference algorithms, and still provides satisfying network based game play experience to players.

5.2.4 The Disadvantages of the LSHBIA

Similar to other algorithms, the LSHBIA works well when the network latency is less than 600ms. However, when the network latency exceeds 600ms, the remote vehicle's movement can not be interpolated and predicted correctly. The remote vehicle's movement may become strange when the source vehicle changes its status suddenly or there are sharp turns during the racing. Such as vehicle flying and vehicle shifting.

The vehicle flying problem is shown in Figure 5.9. In the Figure 5.9, the left one is the remote vehicle and the right one is the source vehicle. We can clearly see that the remote vehicle is flying, which is not a realistic vehicle movement. This happens when the source vehicle moves across a hump as shown in Figure 5.10. The remote vehicle predicts the future movement with the updated moving up status of the source vehicle until the moving down status is received. Due to the large network latency, the moving down information is received after a certain amount of time (600ms). The prediction based on the moving up status previously received will make the vehicle continue moving up instead of moving down.

Figure 5.9 The vehicle flying problem

Figure 5.10 The terrain causes the vehicle flying problem at large network latency

The vehicle shifting problem happens when the source vehicle takes a sharp turn. It is shown in Figure 5.11, where the left one is the remote vehicle and the

right one is the source vehicle. We can see that the remote vehicle is moving out of track and shifting while the source vehicle is still on the track. The reason can be illustrated with Figure 5.12. When the source vehicle has a sharp turn like the one shown in the figure, the remote vehicle does not received such information immediately due to the latency, the remote vehicle predicts its future movement with the previous status of the source vehicle. The remote vehicle follows the predicted path shown in the figure. After the vehicle running out of track, the new movement status notified the turning from the source vehicle may be received. At this moment, the remote vehicle starts to shift back to the correct position through the shifting back path shown in the figure. Figure 5.12.

Figure 5.11 The vehicle shifting problem

Figure 5.12 The reason for the shifting problem when network latency is large

5.3 Experiment results of AI enhanced LSHBIA and analysis

The LSHBIA solves the smoothness problem when the network latency is not large, but when the network latency is large, smoothness can not be guaranteed. Problems like vehicle flying and vehicle shifting may happen. The AI enhanced LSHBIA is used to solve those problems.

To test the smoothness performance of the AI enhanced LSHBIA, the same testing method is used as the LSHBIA. Its performance is compared with the LSHBIA. The comparison result for the both algorithms with 1000ms latency is shown in Figure 5.13.

Figure 5.13 Results of smoothness testing for 1000ms network latency

From Figure 5.13, we can clearly see that the performance of the AI enhance LSHBIA is much better than the LSHBIA when the latency is 1000ms. This is because when the network latency is greater than 600ms, AI instead of LSHBIA is used to control the remote vehicle.

The consistency performance is tested through the same testing method as the LSHBIA. The consistency performance is compared with the LSHBIA and the comparison result with 1000ms latency is shown in Figure 5.14.

Figure 5.14 Results of consistency testing for 1000ms network latency

Form Figure 5.14, we can see that the consistency problem does not improve. This is because the remote vehicle's movement does not use the updated movement information received from the source vehicle to control the remote vehicle's movement when the latency is too large.

To test the performance of the AI enhanced LSHBIA, we set AI techniques to control the vehicle, and the AI levels for the source vehicle and remote vehicle are chosen from the three different levels to test the performance of the algorithm. For each case, the algorithm is tested with 20 times. The smoothness and consistency performance testing methods are the same as the testing methods of LSHBIA. The testing results are compared with the testing results of LSHBIA under same network condition. The latency is chosen to be larger than 600ms, since the AI enhanced LSHBIA uses the same method as LSHBIA to control the remote vehicle's movement when the latency is less

than 600ms, therefore get the same performance. The comparison results are shown in Table 5.1. In the table, the percentage value means how many percentages that the AI enhanced LSHBIA has better performance than the LSHBIA in the 20 experiments.

AI for source vehicle	AI for remote vehicle	Smoothness performance	Consistency performance
Beginner AI	Beginner AI	100%	60%
Beginner AI	Normal AI	100%	20%
Beginner AI	Professional AI	100%	0%
Normal AI	Beginner AI	100%	15%
Normal AI	Normal AI	100%	65%
Normal AI	Professional AI	100%	30%
Professional AI	Beginner AI	100%	0%
Professional AI	Normal AI	100%	25%
Professional AI	Professional AI	100%	70%

Table 5.1 Performance of AI enhanced LSHBIA with latency greater than 600ms

From Table 5.1 we can see that the smoothness performance of the AI enhanced LSHBIA is always better than the LSHBIA. The reason is because the AI enhanced LSHBIA uses AI to control the remote vehicle's movement when the network latency is larger than 600ms.

From Table 5.1, we can also see that the consistency performance varies with the AI levels chosen for the source vehicle and remote vehicle. When the AI levels for the source vehicle and remote vehicle are different, the consistency

performance of the AI enhanced LSHBIA is worse than that of the LSHBIA in most situations. The reason is that when the source vehicle's AI and remote vehicle's AI are different, the movement statuses of them are different also, since the remote vehicle is controlled by the AI, the difference can not be corrected until the latency is less than 600ms.

From Table 5.1, we can also see that when the AI levels chosen for the source vehicle and remote vehicle are the same, the performance of the AI enhanced LSHBIA is better than that of LSHBIA in most cases. That is because both of the source vehicle and remote vehicle have similar movement statuses since they are controlled with the same levels of AI. But there are still cases available, where AI enhanced LSHBIA has worse consistency performance than LSHBIA. The reason is that collision or out of track situation may happen on the source vehicle's side or the remote vehicle's side. If the situations happen only on one side, the other side may not know the situations, this causes the inconsistency between movement statuses of the two vehicles.

The above experiments are conducted under more than 600ms network latency. The network latency may vary around 600ms, sometimes less than 600ms, and sometimes exceeds 600ms. Testing is also done under this situation. The smoothness and consistency performance testing methods are the same as the testing methods of LSHBIA. For each pair of AI case, 20 experiments are conducted. The performance of AI enhanced LSHBIA under this situation is also compared with the LSHBIA. The comparison results are also shown in Table 5.2.

AI for source vehicle	AI for remote vehicle	Smoothness performance	Consistency performance
Beginner AI	Beginner AI	80%	70%
Beginner AI	Normal AI	70%	55%
Beginner AI	Professional AI	65%	40%
Normal AI	Beginner AI	75%	40%
Normal AI	Normal AI	85%	75%
Normal AI	Professional AI	75%	55%
Professional AI	Beginner AI	65%	45%
Professional AI	Normal AI	70%	60%
Professional AI	Professional AI	95%	90%

Table 5.2 Performance of AI enhanced LSHBIA under varied network conditions

From Table 5.2, we can see that the smoothness performance of AI enhanced LSHBIA drops for most of the cases when the network latency varies around 600ms. The reason is that when the latency is larger than 600ms, AI is used to control the remote vehicle. The difference between the status of the source vehicle and status of remote vehicle may be large. When the latency becomes less than 600ms, LSHBIA is used to control the remote vehicle instead of AI. The latest status of the source vehicle can be received by the remote vehicle, and correction needs to be done on the movement of the remote vehicle, the correction from the local status to the received update status causes the un-smoothness performance.

From Table 5.2, we can also see that the consistency performance of AI enhanced LSHBIA becomes better when the network latency varies around 600ms for most of the cases. The reason is when the latency is less than 600ms, the status of the source vehicle can be received by the remote vehicle and remote vehicle can update its status to ensure consistency between the source vehicle's movement and remote vehicle's movement.

In summary, the smoothness performance of AI enhanced LSHBIA is improved compared with the LSHBIA when the network latency is larger than 600ms. However, the consistency performance can not be ensured when the network latency is larger than 600ms.

5.3.2 Advantages of AI enhanced LSHBIA

The AI enhanced LSHBIA provides smooth vehicle movement when the network latency is large. The AI enhanced LSHBIA avoids the problem, like vehicle flying, vehicle shifting, etc. That is because the movement of the remote vehicle is controlled by the AI technique, which can navigate the remote vehicle's movement on the track frame by frame smoothly and is not affected by bad network conditions. The AI enhanced LSHBIA only uses AI when the network latency is large. When the network latency is small, the normal LSHBIA is used. With this, the consistency between the source vehicle and remote vehicle can be ensured when the latency is small.

5.3.3 Disadvantages of AI enhanced LSHBIA

Although the smoothness of the remote vehicle's movement can be guaranteed with this algorithm for most cases, the consistency of the source vehicle and remote vehicle can not be ensured if the wrong AI level is chosen for the remote vehicle. The collision and out of track situation may also affect the performance of the algorithm.

The sudden and unsmooth movement of the remote vehicle may happen if the network latency drops below 600ms after a long time with network latency greater than 600ms. Under this situation, the smoothness of the remote vehicle's movement can not be maintained well. But it still provides better smoothness performance than the LSHBIA.

5.4 Chapter summary

The LSHBIA and AI enhanced LSHBIA are tested under different kinds of network conditions simulated with WANem tool. The LSHBIA provides smoother remote vehicle movement without losing much consistency when the network latency is less than 600ms. But it can not provide smooth movement nor avoid the vehicle flying and shifting problem in some cases when the latency is larger than 600ms. The AI enhanced LSHBIA solves the smoothness problems LSHBIA can not solve when the latency is larger than 600ms.

Chapter 6 Conclusion

This chapter presents conclusions of the research. It also explains some improvements that can be made for future research.

6.1 Summary

This thesis has shown that it is possible to provide smooth remote vehicle movements in network based racing games under most network conditions. We proposed two algorithms called LSHBIA and AI enhanced LSHBIA. Players can hardly see any jitters or unrealistic vehicle movements during their game play with these two algorithms, which also provide satisfying consistency when the latency is small.

LSHBIA is suitable for good network conditions where the latency seldom goes beyond 600ms. Under bad network conditions with average latency to be more than 600ms, the vehicle movements will become unsmooth using LSHBIA. We therefore suggest to use AI enhanced LSHBIA in such network condition. The algorithm will handle over the control of remote vehicles to AI, therefore still maintain good smoothness of vehicle movements even with very high latency. Such strategy can still provide players with satisfying network based game experience. Of course, under such high latency condition, the consistency is not guaranteed using AI enhanced LSHBIA. However, other methods will also have similar inconsistency problems under such circumstance.

6.2 Limitations and Future work

LSHBIA and AI enhanced LSHBIA still have some limitations.

Firstly, both algorithms are mainly designed for network based racing games. A small sized data package containing the position, orientation of the vehicle and time is sufficient to interpolate and predict the future movements of vehicles. However, there are some network based games involving more complex and fast-changing movement behaviors, such as sports games and action games. In these games a large sized data package containing more data is required. However, the data package with the increasing size from various players over the network will easily use up the network bandwidth, although the two algorithms is used. This limitation leaves some room for the future improvement.

Secondly, in AI enhanced LSHBIA, the AI used is chosen from three predefined AI settings which stand for beginner, normal and professional. This is to match the network player's skill based on his performance in history. However, these pre-defined AI solutions are not able to represent players' skill or driving habits accurately. A better solution is to adopt some learning AI algorithms such as neural network. Applying training based on player's performance in history for a specific session with these algorithms can have a more accurate simulation of the player. This can help to improve the consistency when the latency is large.

References

- [1] D. Sanghi, A. Agrawala, I. Gudmundsson, and B. Jain, "Experimental assessment of end-to-end behavior on internet," University of Maryland at College Park 1992.
- [2] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," in *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference* San Diego, CA: USENIX Association, 1996.
- [3] D. Mosberger, L. P. Larry, G. B. Patrick, and O. M. Sean, "Analysis of techniques to improve protocol processing latency," in *Conference proceedings on Applications, technologies, architectures, and protocols for computer communications* Palo Alto, California, United States: ACM, 1996.
- [4] Y. Ishibashi, Y. Hashimoto, T. Ikedo, and S. Sugawara, "Adaptive Δ -causality control with adaptive dead-reckoning in networked games," in *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* Melbourne, Australia: ACM, 2007.
- [5] Y. Zhang, L. Chen, and G. Chen, "Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* Singapore: ACM, 2006.
- [6] L. Pantel and C. W. Lars, "On the suitability of dead reckoning schemes for games," in *Proceedings of the 1st workshop on Network and system support for games* Braunschweig, Germany: ACM, 2002.
- [7] J. Aronson, "Dead Reckoning: Latency Hiding for Networked Games " 1997. [Online]. Available http://www.gamasutra.com/features/19970919/aronson_01.htm
- [8] W. Cai, F. B. S. Lee, and L. Chen, "An auto-adaptive dead reckoning algorithm for distributed interactive simulation," in *Proceedings of the*

thirteenth workshop on Parallel and distributed simulation Atlanta, Georgia, United States: IEEE Computer Society, 1999.

- [9] S. Li, C. Chen, and L. Li, "A new method for path prediction in network games," *Comput. Entertain.*, vol. 5, pp. 1-12, 2007.
- [10] G. Cheung and T. Sakamoto, "Construction and scheduling of extrapolated parity packets for dead reckoning in network gaming," in *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* Melbourne, Australia: ACM, 2007.
- [11] S. K. Singhal and D. R. Cheriton, "Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality," *Teleoperators and Virtual Environments* vol. 4, pp. 169-193, 1995.
- [12] S. K. Singhal, "Effective remote modeling in large-scale distributed simulation and visualization environments," Stanford University, 1997.
- [13] S. Li and C. Chen, "Interest scheme: a new method for path prediction," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* Singapore: ACM, 2006.
- [14] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, pp. 558-565, 1978.
- [15] F. Mattern, "Virtual Time and Global States of Distributed Systems " 1995.
- [16] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *J. ACM*, vol. 32, pp. 52-78, 1985.
- [17] P. Ramanathan, K. G. Shin, and R. W. Butler, "Fault-Tolerant Clock Synchronization in Distributed Systems," *Computer*, vol. 23, pp. 33-42, 1990.
- [18] B. Simons, J. L. Welch, and N. Lynch, "An overview of clock synchronization," pp. 84 - 96 1990
- [19] Z. B. Simpson, "A Stream-based Time Synchronization Technique For Networked Computer Games." vol. 2009, 2000.[online]. Available <http://www.mine-control.com/zack/timesync/timesync.htm>

- [20] Armitage and G. J., "An experimental estimation of latency sensitivity in multiplayer Quake 3 " in *11th IEEE International Conference on Networks/ICON*, Sydney, New South Wales, Australia, 2003 pp. 137-141.
- [21] C. K.-T, H. P., W. G.-S., H. C.-Y, and L. C.-L., "On the Sensitivity of Online Game Playing Time to Network QoS," in *25th IEEE International Conference on Computer Communications*, Barcelona, Catalunya, Spain, 2006, pp. 1-12.
- [22] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?," *Commun. ACM*, vol. 49, pp. 34-38, 2006.
- [23] C. Mark and C. Kajal, "Latency and player actions in online games," *Commun. ACM*, vol. 49, pp. 40-45, 2006.
- [24] T. Yasui, Y. Ishibashi, and T. Ikeda, "Influences of network latency and packet loss on consistency in networked racing games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* Hawthorne, NY: ACM, 2005.
- [25] H. Chen, L. Chen, and G.-C. Chen, "Effects of local-lag mechanism on task performance in a desktop CVE system," *J. Comput. Sci. Technol.*, vol. 20, pp. 396-401, 2005.
- [26] Y. Ishibashi and S. Tasaka, "Causality and media synchronization control for networked multimedia games: centralized versus distributed," in *Proceedings of the 2nd workshop on Network and system support for games* Redwood City, California: ACM, 2003.
- [27] T. Kanbara, Y. Ishibashi, and S. Tasaka, "Haptic media synchronization control with dead-reckoning in networked virtual environments," in *the 8th World Multi-Conference on Systemics, Cybernetics, and Informatics (SCI)*, 2004, pp. 158-163.
- [28] G. J. P. d. Carpentier and R. Bidarra, "Behavioural Assumption-Based Prediction for High-Latency Hiding in Mobile Games," in *CGAMES' 07* Angouleme, France 2005.
- [29] F. George and Hamza-Lup, "Dynamic shared state maintenance in distributed virtual environments," University of Central Florida, 2004, p. 167.

- [30] G. Biasillo, "Representation a Racetrack for the AI,," in *AI Game Programming Wisdom*: Charles River Media Inc, 2002.
- [31] G. Biasillo, "Racing AI Logic," in *AI Game Programming Wisdom*: Charles River Media Inc, 2002.
- [32] G. Johnson, "Avoiding Dynamic Obstacles and Hazards," in *AI Game Programming Wisdom 2*: Charles River Media Inc, 2002.
- [33] M. Nambiar, H. K. Kalita, D. Mishra, and S. Rane, "WANem." [online]. Available <http://wanem.sourceforge.net/>
- [34] R. Vaidya, T. Giang, and D. Terzopoulos, "NEAT Racing: An Evolutionary Neural Network Implementation," University of California, Los Angeles, 2008.
- [35] B. Shumeet, "Evolution of an artificial neural network based autonomous landvehicle controller," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, pp. 450-463, Jun 1996.
- [36] G. Biasillo, "Training an AI to Race," in *AI Game Programming Wisdom* Charles River Media 2002, pp. 455-459.