Improving Collision Detection in Distributed Virtual Environments by Adaptive Collision Prediction Tracking

Jan Ohlenburg
Fraunhofer Institute Applied Information Technology FIT
Sankt Augustin, Germany
jan.ohlenburg@fit.fraunhofer.de

Abstract

Collision detection for dynamic objects in distributed virtual environments is still an open research topic. The problems of network latency and available network bandwidth prevent exact common solutions. The Consistency–Throughput Tradeoff states that a distributed virtual environment cannot be consistent and highly dynamic at the same time. Remote object visualization is used to extrapolate and predict the movement of remote objects reducing the bandwidth required for good approximations of the remote objects. Few update messages aggravate the effect of network latency for collision detection.

In this paper new approaches extending remote object visualization techniques will be demonstrated to improve the results of collision detection in distributed virtual environments. We will show how this can significantly reduce the approximation errors caused by remote object visualization techniques. This is done by predicting collisions between remote objects and adaptively changing the parameters of these techniques.

1. Introduction

In distributed virtual environments multiple users interact with each other or with world objects in real–time. Usually, these users are represented by an avatar in the virtual world. When navigating through the world or interacting with objects the other users are notified of the user action by visual, sensory or audio feedback, e.g. they see how the avatar of that user walks or interacts with objects. To ensure this awareness, messages have to be exchanged between all users. How these messages are distributed among the users is an important factor and depends on the underlying framework of the distributed virtual environment, e.g. [2][4][14]. These frameworks are different in terms of scalability, network architecture and network communication.

The properties of the frameworks have great impact on how and when messages are distributed among the users. The most important factor for collision detection is network latency. Collision detection is usually performed between a pair of virtual objects. In non-distributed virtual environments the position and the orientation of both objects is known at the time of the collision query. This is not the case in distributed virtual environments. Collision detection between two moving avatars can hardly be performed with the exact positions and the exact orientations of both objects. Due to network latency, all update messages arrive some time after the message has been sent. Therefore, at the time of the collision query the position and the orientation could have changed, since another update message has not arrived yet. A fundamental rule about distributed virtual environment's shared state describes this drawback. It is known as the *Consistency–Throughput Tradeoff* [18]:

> ”It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state.”

The *Consistency–Throughput Tradeoff* states, that a distributed virtual environment cannot support both dynamic behavior and absolute consistency. To proof this statement, two scenarios are described, the first consistent and the other highly dynamic.

In a consistent distributed virtual environment no client may change the state of the environment unless all other clients have agreed to the new state. Suppose, a user wants to change his avatar's position and suffers from 100 ms network latency. It takes at least 100 ms until all other clients have received this request and another 100 ms to send an acknowledgement back to user. 200 ms of round–trip time to change the state of the environment means that no more than 5 consecutive state changes can be processed per second. Interaction is therefore limited to 5 frames per second, which is far away from supporting dynamic behavior.

In a highly dynamic distributed virtual environment users may change the state of objects without the agreement of other users. Inconsistencies are handled after they have occurred. Suppose, a user is walking around in the environment, at each frame he sends update messages to all other users about his current position, e.g. current position is p. At the time this update message arrives at the other clients, these clients only know that the user is somewhere near p, since in the mean time his position could have been changed again. As long as users are changing the state of the environment, no consistent state will be reached.

Section 2 provides an overview about the previous work. The following section, Section 3, explains the experiment which has been used to demonstrate the new approaches for collision detection in distributed virtual environments and approaches for remote object visualization, Sections 4 and 5. Additionally, the topic of decision resolution will be discussed in Section 6. At the end of the paper the work will be concluded and an outlook into future research topics is given in Section 7.

2. Previous Work

The distributed virtual environment NPSNET [19] which has the focus on human–computer interaction and software technology for implementing large–scale virtual environments is specialized in military maneuver simulation and has a simple and limited collision detection and collision response. The collision detection process is divided into three levels. In the first level the object's height above the terrain is compared to the maximum height of all objects within close proximity. If the object is not higher than the tallest objects, the process continues with the second level. Here the distances between the objects are computed. If the distance is smaller than the cylinder radii, then a collision has occurred. In this case the process moves on to the third phase — the resolution phase — where the collision response is computed, the collision response is limited to the situation that objects involved either stop or die [15]. The NPSNET Research Group has not implemented the collision detection especially for distributed virtual environments, that is to say, no special strategy has been applied to handling problems encountered in distributed collision detection. But, as many other distributed virtual environments, it utilizes dead reckoning which is a technique for remote object visualization.

Sandoz and Sharkey [16] are discussing the problem of network latency and provide an example where network latency has no side effects. In this example — a distributed snooker game — only the event that the cue ball has been hit by a player is sent. Every participant performs collision detection locally and calculates the positions of the snooker balls independently.

Similarly, in a virtual tennis match [11][12] physical laws can be utilized to predict the movement of the tennis ball. The ball's position can be exactly predicted in spite of high latencies. The user, who has to return the ball, can determine the position of the ball exactly and additionally he knows the exact positions of his racket as well. Therefore he is able to perform exact collision detection.

These two approaches apply to all object movement following strict rules (usually physical laws). For unpredictable movement, e.g. a squirrel running around, other techniques have to be utilized. The following approaches are able to predict the movement of object by extrapolating the movement of the past.

Dead reckoning, used by NPSNET [15], predicts the position and orientation of remote objects by using the last known position and orientation, together with the positional velocity and angular velocity to calculate the actual position and orientation. Second–order dead reckoning also uses the positional and angular accelerations. The same dead reckoning algorithm is executed at the local and the remote host for each dead reckoned object. Therefore, the local host knows the approximation error of the remote host. Two thresholds are used to decide whether to send a new update message to the remote host: an error–based threshold and a time–based threshold. If the approximation error exceeds the error–based threshold, a new update message is sent to the remote host, while the time–based threshold specifies the maximal time between two consecutive update messages. Usually both thresholds are used at the same time. On the arrival of a new update message, the dead reckoning process has to decide whether to *jump* to the actual position, or to calculate a smooth path to a predicted position in the near future. The *smooth back* method reduces visual inconsistencies, but this can result in the object chasing its correct location. By utilizing dead reckoning the network traffic can be reduced by two orders of magnitude, while still having good approximations of remote objects.

A more complex solution to extrapolating the positions and orientations of remote objects is the *position history–based protocol* proposed by Singhal and Cheriton [17]. Like dead reckoning, the local host executes the tracking algorithm as well to be aware of the approximation error. Instead of using the object's positional and angular velocities and accelerations, the tracking algorithm uses the last three positions and orientations of an object. The algorithm adapts to the object's behavior by differentiating between sharp turns and smooth motion. A sharp turn in the object's movement is recognized by calculating the angle between the three most recent update positions. If the angle is small, a sharp turn is assumed; otherwise smooth motion is assumed. In the case of a sharp turn, the two most recent positions are used to extrapolate subsequent positions. In the case of a smooth motion, second–order tracking is applied. The convergence point is adaptively determined depending on the movement, and also the convergence path, which is either parabolic (second–order) or linear (first–order). First–order convergence is applied if the angle between the three most recent positions is large, indicating almost linear movement; otherwise second–order convergence is utilized. Singhal and Cheriton have compared their approach with dead reckoning used by NPSNET. They observed that the position history–based protocol requires less bandwidth in performing at least as well as dead reckoning for smooth object motion and potentially better for non–smooth motion.

Multiplayer Network Games normally utilize centralized architectures, where the centralized server performs all state changes. The client's game state logic only predicts the result of the server. More information about multiplayer network games and how they deal with the problems of network latency can be found in [1].

Object	**scene**	**remote**	**enslaved**
scene	none	Remote Collision Prediction	none
remote	Remote Collision Prediction	Remote Collision Prediction	Adaptive Collision Prediction Tracking
enslaved	none	Adaptive Collision Prediction Tracking	none

Table 1. Overview of applied approaches for distributed collision detection.

3. Experimental Outline

The following approaches apply mainly to non-centralized network architectures. However, they apply to Client/Server network architectures as well, if the server is used only to distribute the messages of the clients; but does not perform collision detection. Additionally, these approaches do not rely on a special collision detection system, they are able to decide which host has to perform the collision query and improve the collision detection results by reducing the approximation errors of the involved objects. For detailed information about collision detection systems see [3][5][8][7][9][10].

Virtual objects of a distributed virtual environment are divided into **scene** objects, **enslaved** objects and **remote** objects. Scene objects are static or their movement is predefined, i.e. each host knows their exact position and orientation at any time. Enslaved objects are moved by the local user (e.g. the avatar), update messages have to be sent to all other hosts to ensure, that they become aware of the movement. Remote objects are enslaved on a remote host (e.g. the avatar of another participant). The position and orientation of remote objects is only approximated because of remote object visualization techniques and network latency.

The interesting object pair collisions are scene–remote, remote–remote and remote–enslaved, because the position and the orientation of at least one involved object is approximated.

Table 1 shows the applied approaches for the different object pairs; *none* means that no approach is applied, only collision detection is executed.

If collisions have to be detected between scene and remote objects or between two remote objects, the local client is not included in the decision whether a collision has occurred or not. Until the message arrives indicating a collision or a change of direction, collisions of remote objects should be predicted. This is done by the method of *remote collision prediction* (see Section 4).

The hardest challenge is the detection of collisions between remote and enslaved objects. No client has exact information about all objects involved. To reduce the approximation errors induced by dead reckoning, the approach *adaptive collision prediction tracking* is utilized (see Section 5).

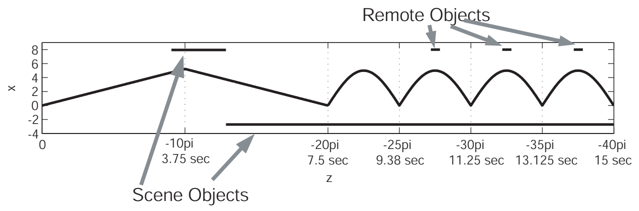

Figure 1. Test scenario

For the following experiments the dead reckoning technique has been used. The results can be applied to the position history–based protocol as well, since they basically work the same way. The main difference would be that the position history–based protocol approximates smooth motions better than dead reckoning.

3.1. The Test Scenario

To demonstrate the effects and the efficiency of the two approaches described below, the following test scenario has been chosen, see Figure 1.

An enslaved object is moved from $z = 0$ to $z = -40\pi$ in 15 seconds. The value for x is determined by Equation 1 and $y = 0$.

$$x = \begin{cases} \frac{-z}{6} & ; \quad |z| < 10\pi \\ \frac{z+20\pi}{6} & ; \quad 10\pi < |z| \leq 20\pi \\ |5\sin(\frac{z}{5})| & ; \quad 20\pi < |z| \end{cases} \qquad (1)$$

Collision with scene objects occur at $z = 10\pi$ and $z = (20 + 5i)\pi$, where $i = 0 \ldots 4$. Remote objects are located at $(8, 0, 27.5\pi)$, $(8, 0, 32.5\pi)$ and $(8, 0, 37.5\pi)$, but no collisions with these remote objects occur. The enslaved object is 5.42m wide.

The test scenario combines smooth motion and sharp turns, which are two common motion classes [17]. The third motion class is random motion. Since tracking and convergence algorithms are of little benefit for this kind of motion, this motion class is not tested with this scenario.

The reason, why random motion is not examined here, is that this type of motion is usually not the result of a collision. More likely, it is the result of collision avoidance, e.g. while someone walks through a crowd of people he will try to avoid running into someone else by changing direction, velocity and acceleration very often. Nonetheless, the two approaches, described below, apply to random motion as well, since they do not make assumptions about the type of motion. While remote collision prediction will be of little use for random motion, the adaptive collision prediction tracking will have comparable results, since the area of interest can be made large enough so that more update messages are able to approximate the motion.

In Section 6 the problem of decision resolution is outlined and exemplary solutions are provided.

All diagrams of the experiments for this test scenario can be found in [13], where the results of the experiments have been provided for different latencies.

4. Remote Collision Prediction

Dead reckoning has its weaknesses, after a collision has occurred, it takes some time for this update message to arrive at remote hosts. Sharp turns, e.g. as a result of a collision, cannot be predicted by the dead reckoning technique, see Figure 2. Therefore, a client has to predict collisions of a remote object on bases of the current approximated object position.

The approach is straight forward. As stated before, if the local client is not included in the collision detection between two objects, because both are remote or one is remote and the other is static, remote collision prediction should be utilized. Collision detection between these two objects is performed with their approximated positions and velocities, but the result is not distributed to other hosts. If a collision occurs an approximated collision response is calculated and used until the next update message arrives.

By applying remote collision predicting, the worst approximation errors have been eliminated. The collision at time $t = 3.75$ is predicted correctly and the approximation error is independent of network latency. The remaining approximation error is due to the different frame rates of the two clients, collisions are detected at different times and therefore with different object positions. The error is negligible, since the predicted path and the visual appearance are correct. Even though the collision response for the collision at time $t = 7.5$ is wrong, remote collision prediction is able to reduce the approximation error by more than 20% and at the time the next update message arrives, the predicted position is less error prone. In Figure 3 the results of the experiments, where remote collision prediction was being applied, can be seen at $t = 3.75$, $t = 7.5$, $t = 9.38$, $t = 11.25$ and $t = 13.125$.

The first two collisions could have been detected with correct remote object positions, because the remote object's movement had been straight. This is not the case for the collisions at time $t = 9.38$, $t = 11.25$ and $t = 13.125$. As the diagrams show, the remote object's position is error prone at the time of collision. In the test scenario remote collision prediction is able to reduce the approximation error to a minimum because the collision response is the same on both clients. Of course, this approach would fail if no collision would have been detected at the remote host, in this case the path would have stayed the same until the next update message arrives.

The most important benefit of remote collision prediction is that it prevents remote objects from entering other objects.

5. Adaptive Collision Prediction Tracking

The reduction of network traffic is achieved by using an error–based threshold, as mentioned before, update messages for enslaved objects are only sent if the error of the predicted position exceeds the error–based threshold. This mechanism implies larger error of remote object positions; the minimal error only depends on network latency. Since the network bandwidth is a limited resource, a trade–off has to be found between high update rates to support collision detection and low network traffic to save bandwidth.

The following approach is able to compensate low update rates at times where collisions between remote and enslaved objects are very unlikely for the near future. This is done by calculating the time of collision for two moving objects. If this time is below a threshold, the parameters for the dead reckoning protocol are adapted to achieve higher update rates, i.e. the error–based threshold is reduced to a small value.

Definition 1 *Two objects are in close proximity if their enclosing bounding spheres intersect, or if they will intersect*

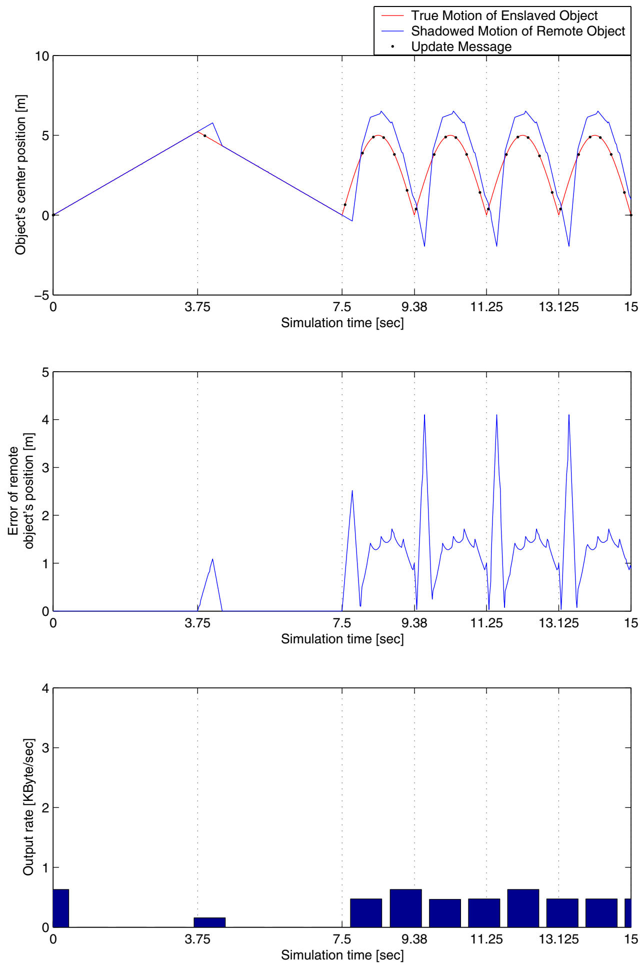

Figure 2. Test scenario at 200 ms network latency with dead reckoning.

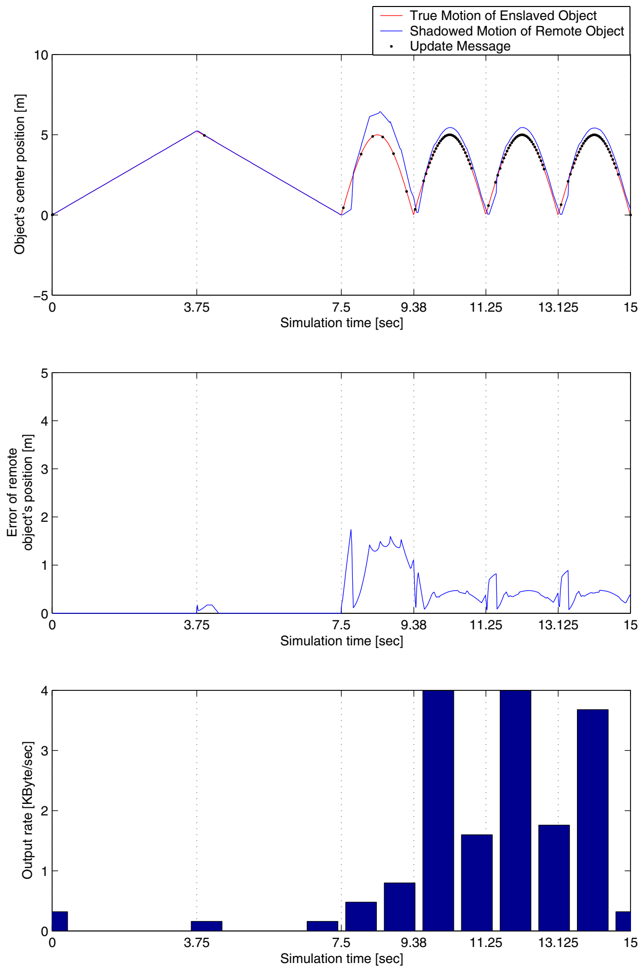

Figure 3. Test scenario with remote collision prediction and adaptive collision prediction tracking at 200 ms latency.

or have intersected within a certain time Δ_τ. Δ_τ is called close proximity threshold.

To test whether two objects, P and Q, are in close proximity, it is checked whether their bounding spheres intersect at time $t = 0$. If they do, no further calculations have to be performed. Otherwise, the time at which the bounding spheres have contact is calculated with their current position and current velocity. The position of object P and Q at time $t = \delta$ is calculated by Equation 2 and Equation 3 respectively. The distance of both objects at time $t = \delta$ is the length of the vector $\overrightarrow{d_\delta}$ (Equation 4), which is calculated by Equation 5. The bounding sphere is used for this approach, since it is the only bounding volume, which is rotation invariant. Due to this property, the distance of two bounding spheres can be described by a closed formula. All other bounding volumes cannot be used for this approach. The disadvantage of the bounding sphere that it fits objects very poorly is an issue for this approach, e.g. a wall 10 m high, 1 m thick and 100 m long has a bounding sphere with a radius of 50.25 m. No matter which direction an object is moving towards this wall the close proximity starts at latest when entering the sphere, see Figure 4.

$$\overrightarrow{P_\delta} = \overrightarrow{P_0} + \overrightarrow{v_P} \cdot \delta \qquad (2)$$

$$\overrightarrow{Q_\delta} = \overrightarrow{Q_0} + \overrightarrow{v_Q} \cdot \delta \qquad (3)$$

$$\overrightarrow{d_\delta} = \overrightarrow{Q_0} - \overrightarrow{P_0} \qquad (4)$$

$$distance_\delta = \sqrt{\overrightarrow{d_\delta} \cdot \overrightarrow{d_\delta}} \qquad (5)$$

$$\begin{aligned} distance_\delta = r \quad &\Leftrightarrow \\ {distance_\delta}^2 = r^2 \quad &\Leftrightarrow \\ \overrightarrow{d_\delta} \cdot \overrightarrow{d_\delta} = r^2 & \end{aligned} \tag{6}$$

To determine the time where the bounding spheres of object P and Q have contact, Equation 5 is set equal to the sum of both radii, r. The calculation is simplified by squaring both sides of the equation, i.e. squared distance equal to r^2.

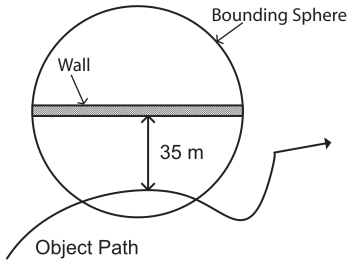

Figure 4. 2D view of close proximity example. Although the object path is far away from the wall, this is a close proximity situation, due to the poor approximation of the bounding sphere.

Solving Equation 6 yields either one, two or no result for δ. No result means, the bounding spheres do not intersect at their current velocities, neither in the past nor in the future. Otherwise, there is at least one result. Let δ_{min} be the minimum of the two results and δ_{max} the maximum, in the case of one result $\delta_{min} = \delta_{max}$. δ_{min} and δ_{max} can both be negative, a negative value indicates that the time of contact was in the past. If δ_{min} or δ_{max} is negative the double of the absolute value is assigned to them. This approach declares two objects within close proximity even if the time of contact was in the past. The reason for that is explained by an example of the test scenario. In the test scenario, the enslaved object avoids collisions with objects at time $t = 10.315$, $t = 12.185$ and $t = 14.06$, after the enslaved object has reached its turning–point and moves into the opposite direction the collision has been avoided. Depending on network latency, the object position at the remote client has not reached its turning–point yet. To ensure that the remote object follows the path of the enslaved object smoothly, the adaptive collision prediction tracking requires for high update rate some time after a collision had been avoided. Since the movement towards another object is more critical than the movement away, negative values are multiplied by -2, i.e. half of the close proximity threshold.

The two objects are in close proximity if the value of $min(\delta_{min}, \delta_{max})$ is less than the close proximity threshold, τ.

The efficiency of this approach can be spotted in Figure 3, where the approaches of remote collision prediction as well as adaptive collision predicting tracking have been applied. At time $t = 8.44$ no obstacle has to be avoided, therefore this approach does not have to be applied. At times $t = 10.315$, $t = 12.185$ and $t = 14.06$ the approach notices that the enslaved object comes into close proximity and uses a different error–based threshold. The resulting average error is significantly smaller than without applying adaptive collision prediction tracking. Of course the network traffic is increased during close proximity situations, but the achieved improvements justify this extra traffic.

Since an application can not only choose the error–based and time–based threshold to find a trade–off between network traffic and exact remote object visualization for each dead reckoned object, but also an error–based threshold for close proximity situations, it is now possible that this trade–off does not aggravate the collision detection results.

6. Decision Resolution

When two enslaved objects collide, it is very likely that the two hosts compute different collision results. As examined in the test scenario, only if one of the objects' movements is straight or does not move at all, the remote side is able to predict its position exactly. The approximation error for turns becomes larger the faster an object moves.

Unless the distributed virtual environment uses a Client/Server architecture, where the collision detection is only performed at the server, there has to be a strategy to decide, which of the hosts' result is chosen to be correct (even if it is not).

This decision has to be made independently on different hosts and the result must be the same. On detecting a collision a host needs to know whether it may distribute the result or not, to prevent the distribution of different messages for the same event. E.g. host A detects a collision of its enslaved object O_A with the remote object O_B enslaved by host B, host A calculates the collision response and distributes the results. Host B has concluded for the same situation that no collision has occurred and therefore will send a message of its new position. Other hosts will not be able to decide which message is correct and will end up in an inconsistent state. The process of choosing a host to be responsible for the collision detection will be named *decision resolution*.

The easiest way to make such a decision is to use the hosts' unique identification number. The host with the lower identification number, A, will be responsible for detection collisions and distributing the results. The other host, B, knows that it has the higher identification number and therefore will not perform collision detection, even if this means that the enslaved object enters the other object's geometry. Since if host A does not detect a collision it will only send update messages of its object and not a message that no collision has occurred. Otherwise, if host B would perform e.g. remote collision prediction this could lead to an inconsistent state, because no resynchronization message for its object is sent. Owing to the simplicity of this method any other criterion will probably produce better results.

6.1. Velocity–Based Decision Resolution

To produce better collision detection results more complex solutions have to be considered. First of all, the maximal approximation error is derived. If no dead reckoning is used or used with an error–based threshold of 0, the maximal approximation error is $dist_{max} = |velocity \cdot latency|$. $dist_{max}$ is the distance an object can move until the message arrives at the remote host at the current velocity. Of course, this implies that the velocity is constant, otherwise the maximal error is $dist_{max} = |velocity \cdot latency + 0.5 \cdot acceleration \cdot latency^2|$. I.e. the higher the latency and/or the velocity is, the bigger the approximation error will be.

This fact can be utilized for deciding which host is responsible for collision detection, since the velocity of remote objects is known. If the host of the object with the higher velocity is chosen the collision detection result will be in general more exact, since the other host will have a higher approximation error for the object. There are some drawbacks of this method, the type of movement is not taken into account and due to network latency, different velocities for the same object can exist at the two hosts. Therefore, if the velocity is chosen as the criterion, a velocity at time $t = t_0 - \Delta_t$ has to be used, where t_0 is the time of the decision and Δ_t has to be greater than any possible network latency. Thus this criterion can only be used for reasonable values of Δ_t. To take the type of motion into account, a position history has to be used, such as used in the position history–based protocol. The usage of Δ_t applies as well.

6.2. Field of View–Based Decision Resolution

One reason, why better collision detection results are desirable, is that visual inconsistencies should be limited. In a distributed virtual environment with low latencies (less than 10 ms) and slow moving objects, e.g. with maximal velocity of 10 m/s then $dist_{max} = 10cm$ and the environment will be resynchronized after a collision within 10 ms. In this scenario the simple approach from above will have the desired effect, tolerable inconsistencies and no noticeable visual inconsistencies. This is not the common case; therefore a criterion for decision resolution can be the reduction of visual inconsistencies. Since collision detection is performed with the information about the enslaved and the remote object of the deciding host, the user will not suffer from visual inconsistencies. Therefore, the field of view of the users can be used. E.g. in a distributed racing game, the player in the car in front, A, only sees the other car of player B through his rear–view mirrors. Collisions with his rear–side and the front–side of the following car should be performed on the basis of the information residing at the host of player B. It also has to be ensured that both host are doing the decision resolution on the same data, therefore the field of view at time $t = t_0 - \Delta_t$ has to be used.

7. Conclusion and Future Work

We showed how the results of collision detection in distributed virtual environments can be significantly improved by utilizing the approaches *Remote Collision Prediction* and *Adaptive Collision Prediction Tracking*. They easily integrate into systems utilizing remote object visualization, including dead reckoning and the position history–based protocol, and the experimental results stated their efficiency. We showed that remote collision prediction can lead to better results of remote object visualization by applying some simple rules. The approach adaptive collision prediction tracking is able to find a trade–off between high update rates to support collision detection and the reduction of network traffic. This was done by distributing update messages at higher rates only if two enslaved objects are in close proximity and a collision becomes more likely. We also showed that the approach of adaptive collision prediction tracking is able to reduce the approximation error to a minimum when required.

Additionally, we outlined the problem of decision resolution and provided criteria to solve this problem.

In our future work, we will be looking into the following research topics.

The collision detection in distributed virtual environments determines the close proximity by utilizing bounding spheres. Poorly approximated objects have a large close proximity area and therefore raise the required bandwidth usage unnecessarily. Subdivision of the object to provide a better approximation of an object to reduce the close proximity area should be very helpful.

Also, collisions between remote objects, which are not in the field of view, do not have to be detected by each client. A client becomes aware of the collision by update messages. Therefore, a strategy to decide for which objects collision detection has to be performed should be added to the system.

Quality of Service (QoS), which is part of the new internet standard IPv6, is able to provide a known upper–bound on latencies and a guaranteed bandwidth [6]. With this service completely new approaches are possible, since these approaches can rely on fixed circumstances and therefore are able to ignore possible network congestions.

References

[1] Y. W. Bernier. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. In *Proceedings of the 15th Games Developers Conference*, 2001.

[2] W. Broll. Smalltool – a toolkit for realizing shared virtual environments on the Internet. *Distributed Systems Engineering*, 5(3):118–128, 1998.

[3] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi. I–COLLIDE: An Interactive and Exact Collision Detection System for Large–Scaled Environments. *In Proc. of ACM Int. 3D Graphics Conference*, pages 189–196, 1995.

[4] E. Frécon and M. Stenius. DIVE: a scalable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(4):91–100, 1998.

[5] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. *In Proc. of ACM Siggraph*, pages 171–180, 1996.

[6] R. M. Hinden. IP next generation overview. *Communications of the ACM*, 39(6):61–71, 1996.

[7] P. M. Hubbard. Collision Detection for Interactive Graphics Applications. *In IEEE Transactions on Visualization and Computer Graphics*, pages 218–230, 1995.

[8] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V–COLLIDE: Accelerated Collision Detection for VRML. *In Proc.* 2^{nd} *Annual Sympos. on Virtual Reality Modeling Language*, 1997.

[9] J. T. Klosowski. *Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments*. PhD thesis, State University of New York at Stony Brook, 1998.

[10] M. Moore and J. Wilhelms. Collision Detection and Response for Computer Animation. *In Proc. of Computer Graphics (SIGGRAPH)*, 22(4):289–298, 1988.

[11] H. Noser, I. S. Pandzic, T. K. Capin, N. M. Thalmann, and D. Thalmann. Playing Games through the Virtual Life Network. In *ALIFE V*, pages 114–121, 1996.

[12] H. Noser, C. Stern, P. Stucki, and D. Thalmann. Generic 3D Ball Animation Model for Networked Interactive VR Environments. In *Virtual Worlds*, pages 77–90, 2000.

[13] J. Ohlenburg. Efficient Collision Detection for Dynamic Objects in Distributed Virtual Environments. Master's thesis, Rheinisch–Westfälische Technische Hochschule Aachen, May 2003.

[14] S. Powers, M. Hinds, and J. Morphett. DEE: an architecture for distributed virtual environment gaming. *Distributed Systems Engineering*, 5(3):107–117, 1998.

[15] D. R. Pratt. *A Software Architecture for the Construction and Management of Real–Time Virtual Worlds*. PhD thesis, Naval Postgraduate School, Monterey, CA, 1993.

[16] P. D. Sandoz and P. M. Sharkey. Collision detection in distributed virtual worlds. 3^{rd} *UK Virtual Reality Special Interest Group Conference*, 1996.

[17] S. K. Singhal and D. R. Cheriton. Using a Position History–Based Protocol for Distributed Object Visualization. Technical Report CS-TR-94-1505, Stanford University, 1994.

[18] S. K. Singhal and M. J. Zyda. *Networked Virtual Environments: Design and Implementation*. ACM Press, 1999.

[19] M. J. Zyda, W. D. Osborne, J. G. Monahan, and D. R. Pratt. NPSNET: Real–Time Vehicle Collisions, Explosions and Terrain Modifications. *The Journal of Visualization and Computer Animation*, 4(1):13–24, 1993.