The Effect Management in Distributed Virtual Environments

Souad Elmerhebi, Patrice Torguet, Nancy Rodriguez, Jean-Pierre Jessel

IRIT - Institut de Recherche en Informatique de Toulouse
118, route de Narbonne
31400 Toulouse
France
merhebi@irit.fr, torguet@irit.fr, rodri@irit.fr, jessel@irit.fr

Abstract. Since the distributed virtual environments are growing in size and in number of participants, the number of update messages that is exchanged between host computers is increasing dramatically, which makes the management of update messages difficult. Many filtering techniques in distributed virtual environments are presented to reduce the number of update messages and thus reduce the load on the hosts and the network to be able to scale in the number of participants. However, filtering update messages may cause a lack of realism because of the loss of significant messages. This paper describes the effect management, which is a filtering technique that seeks to maintain the quality of filtering offered by the existing techniques with respect to the number of exchanged messages while trying to assure a better realism of the application. The effect management associates to each virtual object of the environment an effect zone that delimits a part of the space inside which the object can be perceived by the entities present in the scene. Moreover, it associates to each entity a viewing zone that reflects its visual capacity. When the viewing zone of an entity overlaps the effect zone of an object, the entity will be able to see this object; it starts then to receive the update messages related to it. This technique allows a better realism of the distributed virtual reality applications and a higher scalability.

1 Introduction

The interaction of a participant within a distributed virtual environment produces a change in the state of this environment. To realize a good coherence between the states of the environment on the various hosts, whenever a change occurs at one host, it must send an update message (containing the position and orientation of manipulated objects) to the other hosts to notify them of the change that took place. On the other hand, to ensure the sense of immersion of a user in the virtual world, he must think that he is interacting in real time. Therefore, his host must receive the updates of other hosts quickly enough to give the impression of real time manipulation. As a result, the realism of a distributed virtual reality application depends on the good management of update messages.

Since the distributed virtual environments are growing in size and in number of participants, the number of update messages that circulate between the hosts is increasing dramatically, which makes the management of update messages difficult. This has effects on the hosts and the network:

- A participant is only interested in the part of the virtual environment that is available to him: no participant is actually able to have an effect overall the virtual environment. If its host receives all the messages exchanged within the application, it will have difficulty in treating all of them in real time, which will increase the computational complexity and will affect the rendering speed.
- In addition, this great number of update messages will cause traffic on the network that has a limited bandwidth, which may lead to a loss of messages or to an increase in latency. In fact, if latency becomes high, there will be a delay in the reception of the update messages and this will cause a problem in the realism of the simulated entities' behavior.

These consequences prevent a high scalability related to the number of participants (an increase in the number of participants will cause an increase in the number of update messages) and to their geographic location (that may cause an increase in latency). It is thus necessary to minimize the number of messages exchanged on the network and the number of recipients of these messages without restricting the freedom of action of the participants or the dynamism of the application. For these reasons, it is crucial to integrate filtering in distributed virtual reality applications.

While essential, filtering may eliminate a number of update messages that are useful to the realism of the application. The elimination of such messages might make the user unaware of objects or actions that should be evident in the simulation or it might cause the inexplicable disappearance of important objects in the scene which may perplex user.

This paper presents a new filtering technique in distributed virtual environments called the effect management. This technique seeks to maintain the quality of filtering offered by the existing techniques with respect to the number of exchanged messages while trying to assure a better realism of the application. In section 2, we present a brief overview of related work. Section 3 introduces the basic concept of this management, the effect zone of virtual objects and the viewing zone of entities. Section 4 describes the architecture used to structure the implementation. In section 5, we present the platform used to implement this technique. Section 6 presents conclusions and outlines the remaining work.

2 Related work

Up to now, the filtering techniques that were conceived can be classified into two categories:

- Prediction methods: These methods predict the behaviors of the simulated entities based on the received update messages. An update is only required when the real behavior of the entity deviates significantly from the predicted behavior. Each host calculates the rate of deviation of behavior of its own entities and sends an update message when the rate of deviation reaches a certain threshold. This reduces the number of exchanged update messages. The most known methods of prediction are the dead-reckoning implemented in SIMNET [1] and the explicit model of behavior implemented in WAVES [2].
- Interest management methods: These methods deliver messages to hosts according to the interest of the entities that represent their participants. These methods specify for each entity the data that interests it and send it the update messages concerning these data. Macedonia et al. [3] classified interest filtering as follows:
    - Functional filtering: The entity chooses to communicate with a subset of other entities based on their functionalities.
    - Spatial filtering: The entity interacts with the entities that it considers close to it.
    - Temporal filtering: Some entities do not require real time updates of the state changes of the other entities; thus, they will not receive frequent updates as the remainder of the entities.

  Macedonia et al. implemented spatial filtering in NPSNET [3] by dividing the space into hexagons and specifying for each entity an area of interest, which is a circle centered on the entity. This area of interest delimits a part of the space so that our entity will be able to interact only with the entities located inside this area; these entities are thus considered its interesting entities. This is referred to as the area of interest management. The three-tiered interest management [4] also specifies for each entity an area of interest. The first tier divides the world into dynamic regions, and then subdivides the overpopulated regions to keep only the ones that interest the entity. The second tier eliminates some of the non intersecting entities among those handed by the first tier in a protocol independent manner while the third tier adds protocol dependence allowing an entity to receive only the data from the protocol it needs. HLA [5] also adopted a type of interest management for its filtering based on the services presented by the Runtime Infrastructure: the "Declaration Management" and the "Data Distribution Management". The spatial model of interaction, implemented in MASSIVE [6], is based on the concept of aura. Each object in the virtual world has an aura that delimits a part of the space; this aura defines the extent to which interaction with other objects is possible. When the auras of two objects collide, the interaction between the pair of objects is enabled and therefore they become aware of one another. Storey et al. [7] presented an approach of auras' collision detection that reduces the number of pairwise comparisons and bilateral relations by detecting auras collisions and generating sets of intersecting auras called Collision Relations. A collision relation (CR) is a sphere that encloses overlapping auras. Therefore, an object sends its update messages to the objects of the CRs to which it belongs. Third party objects, implemented in Massive-2 [8], are an extension to the spatial model of interaction. A third party object is an independent object that affects the awareness between other objects. Third parties are used to manipulate awareness levels in nested spatial structures and in mobile crowds.

The majority of the recent methods of filtering adopt the area of interest management. This type of management causes a problem of lack of realism: since the area of interest of an entity delimits the part of the space inside which the entity is able to interact with the virtual world, the entity will thus be isolated inside this area. It will be aware only of its neighbors and all that there is outside its area, though relevant according to its size, does not exist for it.

If we wanted to simulate a field using the area of interest management, not all our entities would be able to see the sun (or any such big but far object) because the sun does not belong to the areas of interest of all the entities (figure 1); besides when the entity moves toward the region where the sun is actually located, the sun will come out suddenly to his view. All

this is not realistic at all. Furthermore, if we wanted to simulate a battle field, a regular soldier may not be able to see a distant tank if the area of interest of the soldier was not wide enough to cover the tank and this is unrealistic and unfair. Even if he came close to it unintentionally, he will see suddenly a near tank in front of him and it would be too late for him to avoid it because it would be already too close. If we tried to solve these problems by enlarging the area of interest of the soldier, the soldier would then be able to see the far tank that may attack him, but the host of the soldier would end up by receiving all the update messages including those that do not interest him.

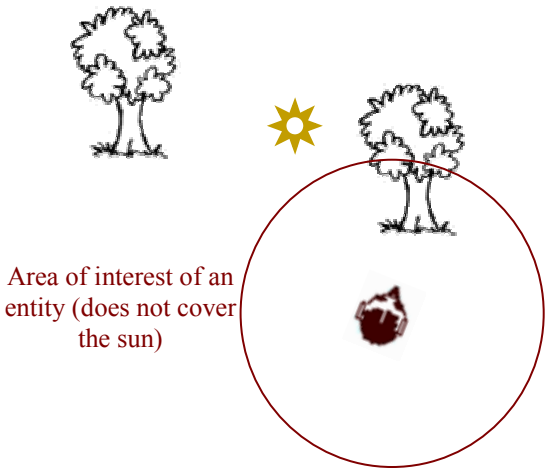

Figure 1 : interest zone

Farcet [9] proposed a space-scale structure (SSS) that stores a hierarchical description of the world. This structure is a dynamic octree: the objects of the world are referenced into this octree according to their position and size. Each object is approximated by its bounding sphere and inserted in the cell whose scale matches the radius of the bounding sphere and whose position suits the location of the object. The octree root cell is dimensioned in order to contain the whole world. To determine which objects are perceptible by each observer, it is necessary to pass through all the occupied cells of the octree several times following the various scales to determine which cells to keep and which cells to reject. Therefore, calculations are carried out for each cell according to its size and according to the distance that separates it from the observer using an approximating rejection function called Projected Area. This method is more realistic but it requires a lot of computation.

To solve the problem of lack of realism caused by the area of interest management without adding a big load of computation, we propose an effect management that associates to each object of the scene an effect zone that reflects its size and consequently its importance in the scene and to each entity (an object that represents a participant) a viewing zone that reflects its capacity of vision. When the viewing zone of an entity overlaps the effect zone of an object, it will begin receiving its updates and therefore will be able to see it.

3 Concept of the management

3.1 Effect zone

We associate to each object a spherical zone, centered on the object and having a ray that reflects its size. This zone reveals the importance of the object in its environment; it is called the effect zone. Only the entities located within the effect zone of an object will be aware of it. Since an entity is also an object and is seen by other entities, it also has its own effect zone. In figure 2, we notice that entity A will see the sun and one tree.

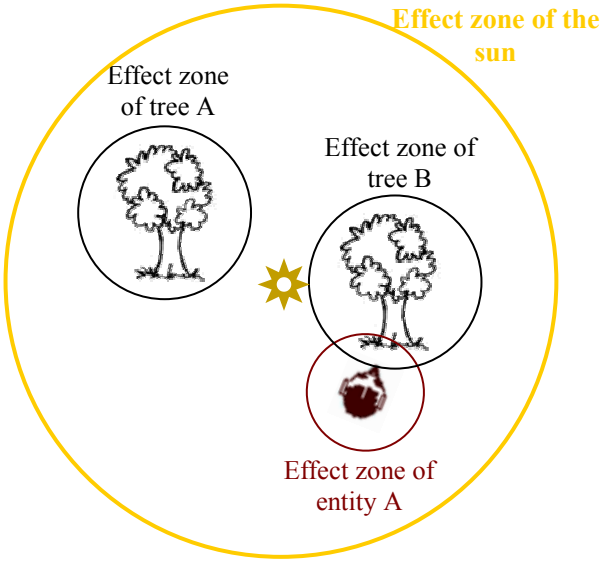

Figure 2 : effect zone

Since the sun of our example has an effect zone that covers the entire field, all the entities will be able to see it. Besides, they will remain unaware of the small and distant objects because their effect zones are restricted and do not cover all the entities of the scene. In this way, we will insure at the same time realism and scalability to the application.

The proximity will not thus remain the only condition that determines the entity's awareness of the objects present in its environment. The individual importance of a virtual object will be a factor in determining whether this object is seen by the entities of the scene.

3.2 Viewing zone

Not all the entities have the same visual capacity: there can be an entity that simulates a shortsighted person or an entity that simulates a person who carries binoculars. It is thus necessary to differentiate between these various types of entities. To this end, we associate to each entity a viewing zone related to its capacity of vision. This zone is also a sphere centered on the entity and has a ray that indicates its capacity of vision. When the viewing zone of an entity overlaps the effect zone of an object, it will be able to receive information concerning this object and will be consequently aware of it. Having a spherical viewing zone spares us the trouble of having new computations whenever the entity changes its orientation. We only have new computations when the entity changes its position not its orientation. In figure 3, entity B is shortsighted it will only see the sun but entity A has a normal view it will see the sun and a tree.

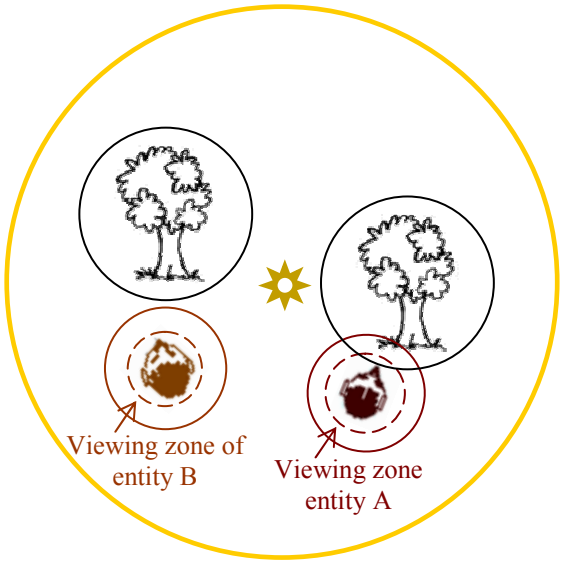

Figure 3 : viewing zone

Therefore, each object has an effect zone and each entity has an effect zone and a viewing zone.

The zones presented in this paper are intended for vision, but we can associate to the same object other types of zones: an audio effect zone, a functional effect zone and to the same entity: an audio zone, a functional zone...

With the previously presented techniques, the ability of an entity to see an object depended on the area of interest (or the aura) of the entity and on the distance that separates the entity from the object. While with the effect management, not only entities have zones, but also objects have their own effect zones, which gives each object an individual importance or impact in the scene. The factor of the effect zone of the object, added to the previous factors (the viewing zone of the entity and the distance that separates it from the object), decide whether or not the entity sees the object. So the individuality of the objects becomes an equally important feature in the filtering procedure. This helps bringing a better realism to the application.

4 Architecture and communication

The client server architecture is used to implement this filtering technique. The server will manage the filtering in the application; it will be the effect manager. It has a database containing the states and characteristics of the objects and the entities (viewing zones, effect zones…). When a user changes the state of his entity, his client sends an update message containing his new state to the server. Consequently, the server makes computations of the overlaps between effect zones and viewing zones to decide what update messages he should send and to whom of his clients. In fact, due to the movement of a user, the server would send different update messages to:

- The user that has moved in case there were:
  - New objects that were not visible and became visible in order that he would add these objects.
  - Old objects that were visible and are not anymore in order that he would remove these objects.
- To the other users in case they were:
  - Not able to see the entity but are now able to see it in order that they add it.
  - Able to see the entity that has moved but are not able to see it anymore in order that they remove it.
  - Able to see it and are still able to see it in order that they update its state.

Multicast has been proven to be a mode of communication that suits well filtering; especially in distributed virtual environment applications since often the same update message is sent to multiple users. So multicast saves the cost of the repetitive transfers of the same message to different users, this seriously reduces traffic network.

On the other hand, having one server may cause a problem of reliability in the application.

4.1 Integration of multicast

We could associate to each object of the scene a multicast group so that when the viewing zone of an entity overlaps the effect zone of an object, the entity's host joins the multicast group of the object. But in this way, there would be a great number of multicast groups especially when the number of objects increases. For this reason, we suggest to divide the space into cells and associate to each cell a multicast group.

4.2 Division of space

After dividing space into cells, each object sends its data to the multicast groups of the cells that overlap its effect zone and each entity joins the groups of the cells that overlap its viewing zone.

In figure 4, the effect zone of the sun overlaps all the cells of the scene, as a result the sun will send its information to all the cells in order for all the entities to see it. Tree B will send its information to cells b2 and b3 because its effect zone overlaps only with these two cells. Entity A will join the multicast groups of cells b2 and b3 since its viewing zone overlaps these two cells but it will also send its information to these two cells because of their overlap with its effect zone. In this way, entity A will receive the information of tree B and the sun as intended.

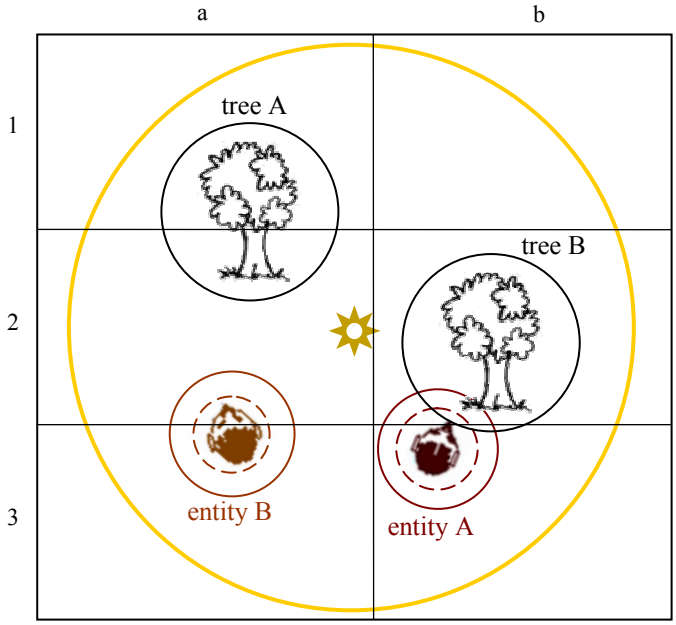

Figure 4 : division of space into cells

The problem of the division of space into cells is the size of the cells. If the cells are small, there will be a great number of multicast groups, and the entities will frequently join and leave multicast groups when they move, especially when they move quickly. In fact, joining and leaving multicast groups is expensive in time when it is done in a wide area network.

If the cells are large, the movement of the entities will not lead to a frequent change of multicast groups. However, the entities will receive data that do not interest them since they join multicast groups that represent large regions. Furthermore, if there are too many objects and entities in the same area, the problem will become increasingly critical.

As a solution, we propose to keep the large cells and to generate dynamic regions consisting of one or many cells according to the number and to the distribution of the objects and entities in the cells. Then we associate to each region a server and a multicast group. In this way, we end up with two layers of filtering:

- An intra-region filtering layer: the server will be responsible for filtering data inside its region. An entity will thus only receive the data that interest it.
- An inter-region filtering layer: servers will communicate between them via multicast, delivering to each another the data that interest their entities.

4.3 Intra-region filtering layer

Each server is in charge of the objects (including the entities) present in its region.

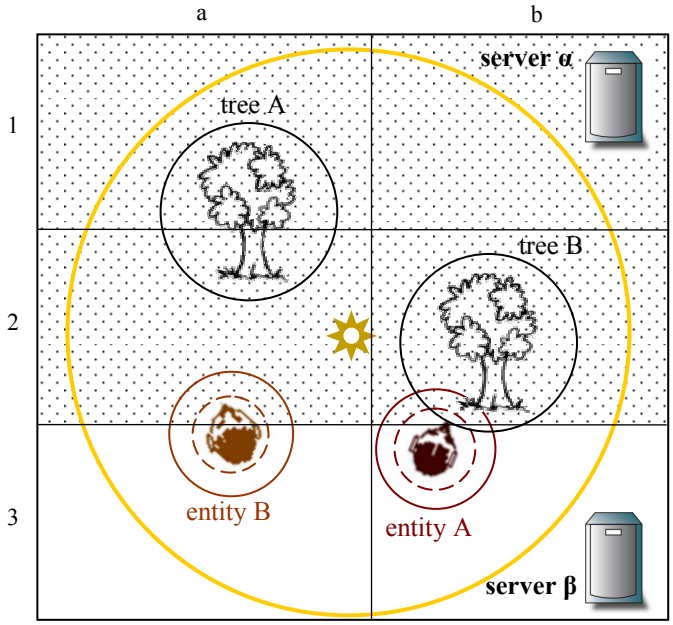

Figure 5 : division of space into regions

In figure 5, the region of server α consists of cells a1, a2, b1 and b2; therefore server α is in charge of tree A, tree B and the sun. On the other hand, the region of server β is composed of cells a3 and b3; server β is thus in charge of entities A and B.

An object generates an update message once its state changes. It sends this update message to its server. Afterwards, the server sends this update message to the interested entities in its region via unicast and to the servers subscribed to its multicast group via multicast (4.3).

In every region, the server goes through all the update messages of the objects that belong to other regions (received via multicast). It routes each message to the entities that belong to him who are interested in the object that generated the message.

As a result, an entity will only receive interesting update messages. For example, when the state of tree B changes, the host of tree B sends an update to server α. Server α then sends the update message to server β given that server β is a member of the multicast group of server α (4.3). When server β receives the update message, it sends it only to entity A since the viewing zone of entity A overlaps the effect zone of tree B while the viewing zone of entity B does not.

4.4 Inter-region filtering layer

Joining and leaving multicast groups is the responsibility of the servers of the application.

Whenever a server realizes that one of his objects' effect zone overlaps another region, it automatically notifies the server of this region. Subsequently, the second server checks if one of its entities is interested in this object. If this is the case, the server will join the multicast group of the first region in order to receive real time update messages regarding this object of interest.

Going back to the scenario described by figure 5, the following actions take place:

- Server α realizes that one of its objects (tree B) has an effect zone that overlaps region β.
- Server α sends server β a notification that contains the position and the radius of the effect zone of tree B.
- Server β checks whether tree B's effect zone overlaps one of its entities effect zone. As a result, it finds out that its entity A is interested in tree B.
- Server β joins server α's multicast group and is thus able to receive real time updates regarding its objects.

The first server will of course stop notifying the second server of new overlaps as long as it is subscribed to its multicast group, because the second server will be already receiving all the update messages of the first region.

Otherwise, after the notification if the server of the second region realizes that none of its entities is interested in the object whose effect zone overlaps its region, it decides not to join the group (which is not the case in our example). Therefore, the first server will send it new notifications when a new overlap takes place or when the state of the object changes, because perhaps the new position of the object would interest one of its entities. However, the notifications of the state's changes will not be sent in real time but in a rather lower frequency.

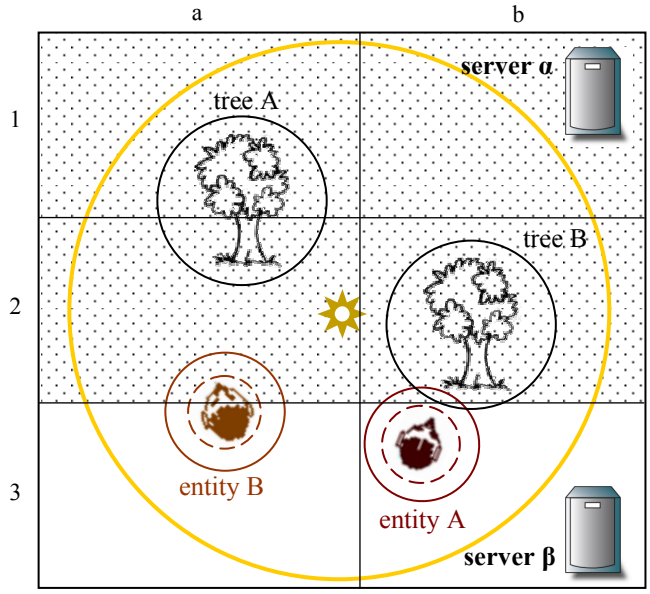

Figure 6 : movement of an entity

After joining a multicast group, when an object or an entity moves in such a manner that there isn't any overlap at all, the server realizes that the update messages received via multicast do not interest it anymore so it leaves the multicast group.

Figure 6 shows that entity A has moved back a little in such a way that its viewing zone no longer overlaps the effect zone of tree B. Server β checks for another overlap and finds out that the sun of region α still overlaps its entities A and B so it can not leave the multicast group of region β.

When an entity or an object leaves a region and enters a new region, the server of the old region sends a notification to the server of the hosting region containing the data of the entity or object. The server of the hosting region takes the entity or object in charge and sends an acquisition to its old region who gives it up.

5 ASSET

The platform that we are using is ASSET (Architecture for Systems of Simulation and Training in Teleoperation) [10], which has been developed by Nancy Rodriguez in our research center.

ASSET is a flexible prototyping system supporting the design and evaluation of new teleoperation systems using virtual reality. This system is a set of reusable Java components. It provides a modular design to easily make extensions and modifications.

5.1 Architecture

The structure of ASSET is composed of a general communication server and two client modules (one in the user side and one in the real system side). As shown in figure 7, the ASSET's architecture consists of:

- User Interface Manager. The User Interface Manager (UIManager) is responsible for the communication between the system and the user; it handles the local simulation and the interaction devices. Communication between this module and the others modules of the system is managed by the Communications and Events component.
- Real System Manager. This module controls the real system. It is very similar to the UIManager, but it controls the sensors and effectors of a real device (e.g. a robot). Real System Manager (RSManager) executes the commands and manages coherence between the real state and the simulated state in order to update the user's simulations.
- Administrator. The administrator coordinates the interactions between the participating entities, users and robots. It is a communication server that transmits the commands to effectors and the information from the real system to the users. It has a Coordination component to solve conflicts raised by orders of different users.

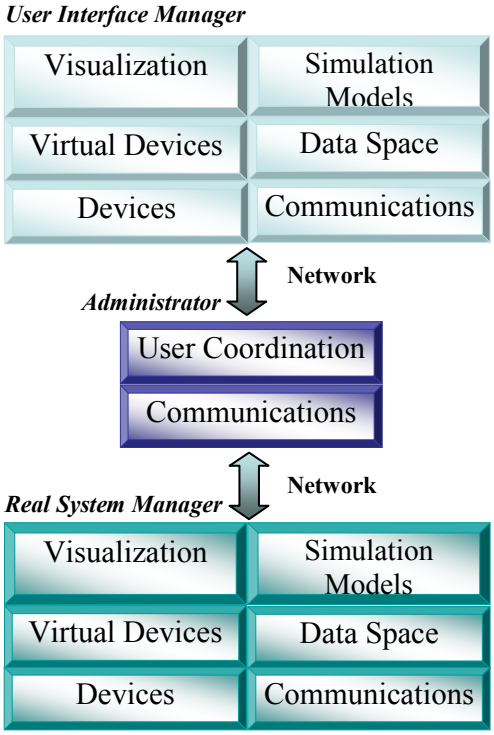

Figure 7 : architecture of the ASSET system

ASSET defines mechanisms that allow the interaction between the different modules and the different components of each module. To set the communication between the various components of a module, a data space is defined to maintain device information, commands, and network messages.

The data space notifies occurrences of a written event (when a component adds a message) allowing devices, simulation objects or communication units to recover and to process it. This capability allows having application domain and devices independence.

5.2 Simulation

The simulation component in ASSET managers offers the services of 3D visualization, collision detection and Java3D or VRML2.0 models loading. ASSET allows defining behavior for each simulation object, so it is possible to have entities with different degrees of autonomy sharing the same environment.

5.3 Implementation

In our context, ASSET is used in simulation mode only -i.e. without communicating with the real world. Therefore, we use only the UIManger and the Administrator.

We added to ASSET the possibility of managing multiple users. These users are aware of each other and have the freedom of participation and of real time interaction in the virtual environment.

We replaced the reliable TCP communication protocol with non-reliable UDP to enhance real time communication because UDP does not require establishing connections and sending acknowledgements that increase the transactions between the hosts and the processing at each host. On the other hand, UDP is required to make the transition to multicast.

We have implemented the area of interest management and the effect management into ASSET and compared between the two techniques with respect to the realism of the application.

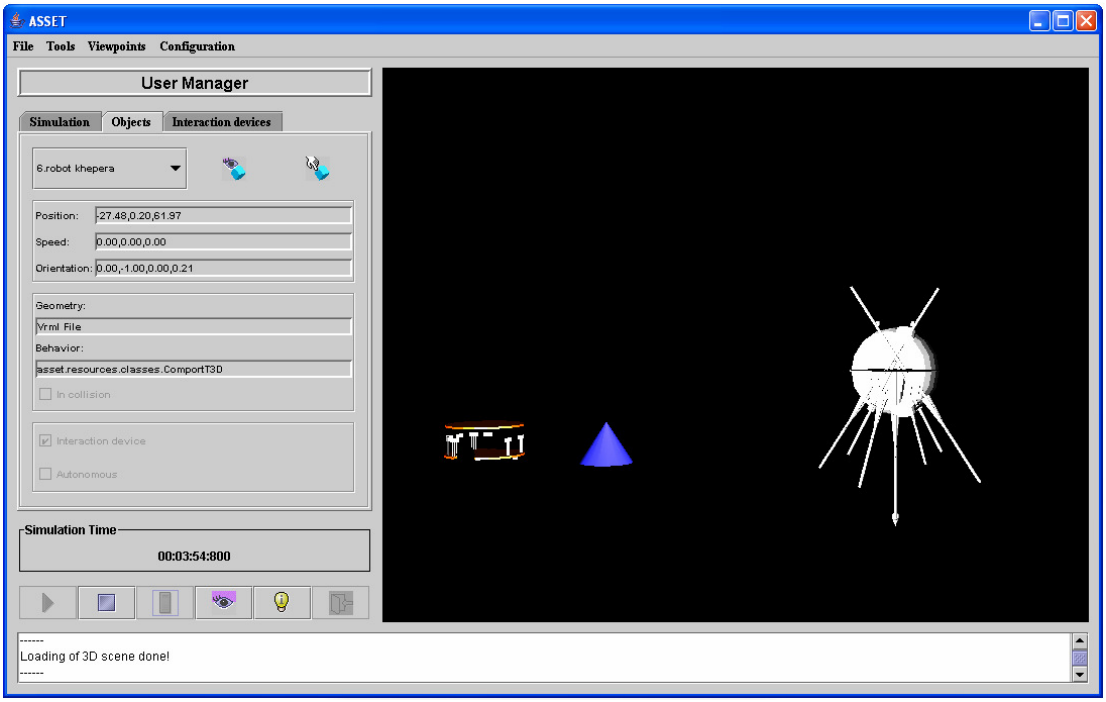

Figure 8: Area of interest management implementation

Figure 8 shows a snapshot from the area of interest implementation into ASSET; our entity has an area of interest that allows us to see a cone, a robot and another entity. These are the only objects included in the area of interest of our entity.

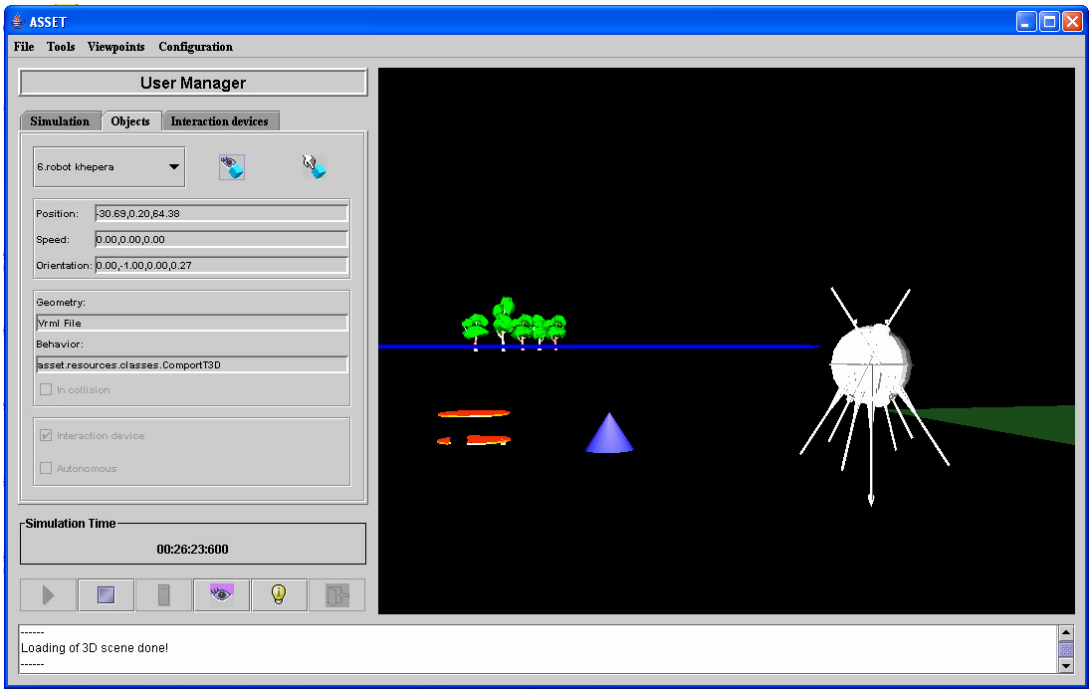

Figure 9: Effect management implementation

Figure 9 shows a snapshot from the effect management implementation, our entity is in the same position as in the prior example but now it has a viewing zone and all the objects of the scene including the entities have effect zones. As we notice, in addition to the near objects seen previously we are now able to see the trees and the field that were not visible before because of their remoteness from our entity. This is due to the collision between our viewing zone and the effect zones of these objects. Furthermore, this management allows us to get rid of the problem of the abrupt appearance and disappearance of the big objects of the scene.

Therefore, we realize that the effect management is much more realistic than the area of interest management, especially if the remote and invisible object is an entity (a plane, a tank…) that we are supposed to perceive and interact with.

6 Conclusions and future work

We have presented the effect management in distributed virtual environments, which is a filtering technique that can be adopted by distributed virtual reality applications to provide a better realism and a better scalability. We have implemented this technique in an already existing virtual reality system.

We will shortly use this implementation in order to evaluate the efficiency of the effect management with respect to the number of exchanged messages and to the computational load and thus the scalability of the application. We would like then to compare the results of our management with those of the area of interest management.

We would like also to replace the server by several servers (called administrators in ASSET) since we may risk a problem of reliability with a single server. We would like to add multicast as a complementing mode of communication such as the servers will communicate between them via multicast but the clients will still communicate with their server via unicast.

It is also required to seek a mechanism that determines the appropriate size of the effect zones and of the viewing zones of the objects and the entities based on the rules of optics. In addition, it is necessary to determine the optimal size of the cells that divide the space and the technique that handles the dynamic regrouping of cells into regions.

6 Acknowledgments

This research is supported by the Lebanese National Council for Scientific Research (CNRS Lebanon).

References

[1] J. O. Calvin, A. Dickens, B. Metzger, D. Miller, D. Owen. The SIMNET Virtual World Architecture, Proceedings of IEEE VRAIS'93, 450-455, 1993.

[2] R. Kazman. Making WAVES: On the Design of Architectures for Low-end Distributed Virtual, Proceeding of the 1st IEEE Conference on Virtual Reality Technology (VRAIS'93), 1993.

[3] M. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham. Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments, Proceedings of the IEEE Virtual Reality Annual International Symposium, (VRAIS '95), 2-10, 1995.

[4] H. Abrams, K. Watsen, M. Zyda. Three-Tiered Interest Management for Large-Scale Virtual Environments, Proceedings of the ACM symposium on virtual reality software and technology, 125-129, 1998.

[5] J. O. Calvin, R. Weatherly. An Introduction to the High Level Architechture (HLA) Runtime Infrastructure (RTI), Proceedings of the 14th Workshop on Standards for the Interoperability of Distributed Simulations, 96-14-103, 1996.

[6] C. Greenhalgh, S. Benford. MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading, Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95), 27-34, 1995.

[7] K. Storey, F. Lu, G. Morgan. Determining Collisions between Moving Spheres for Distributed Virtual Environments, Proceedings of the Computer Graphics International (CGI'04), Crete, 140-147, 2004.

[8] S. Benford, C. Greenhalgh. Introducing Third Party Objects into the Spatial Model of Interaction, Proceedings of the European Conference on Computer Supported Cooperative Work, 189-204, 1997.

[9] N. Farcet, P. Torguet. Space-scale Structure for Information Rejection in Large-scale Distributed Virtual Environments, Proceedings of IEEE Virtual Reality Annual International Symposium 1998 (VRAIS 98), 276-283, 1998.

[10] N. Rodriguez, J. P. Jessel, P. Torguet. Virtual Reality in Cooperative Teleoperation, Poceedings of the 1st Ibero-American Symposium on Computer Graphics, 2002.

[11] J. Gemmell, J. Liebeherr, D.Bassett. An API for Scalable Reliable Multicast, International Conference on Computer Communications and Networks, (Las Vegas, Nevada), IEEE Computer Society, pp. 60-42, 1997.