Scheduling for Very Large Virtual Environments and Networked Games Using Visibility and Priorities

Chris Faisstnauer, Dieter Schmalstieg, Werner Purgathofer
Vienna University of Technology
faisst@cg.tuwien.ac.at

Abstract. *The problem of network bandwidth limitations is encountered in almost any distributed virtual environment or networked game. In a typical client-server setup, where the virtual world is managed by a server and replicated by connected clients which visualize the scene, the server must repeatedly transmit update messages to the clients. By employing visibility information, the number of messages transmitted over the network can be reduced by sending each client only update messages for objects which are visible from the viewpoint of the client. A naïve approach requires to examine visibility between pairs of objects and leads to a quadratic effort in the number of objects, substantially affecting the scalability of such environments. This paper presents a technique that allows the server to manage the transmission of update messages for each client with a constant overhead, reducing overall computational cost to a linear effort. We show how the server can employ visibility information to schedule all objects using the Priority Round-Robin algorithm. This algorithm is further enhanced with activity monitoring that provides a graceful degradation of the system's performance, even if the behavior of objects is unpredictable. This makes the algorithm suited to schedule update messages regarding server-controlled objects as well as user-controlled avatars.*

1 Introduction

In the simulation of large virtual environments, contention for limited resources such as CPU, rendering pipeline, or network bandwidth, frequently causes a degradation of the system's performance. Most distributed virtual environments therefore employ techniques to reduce the number of messages to be transmitted over the network.

A popular approach to build virtual environments, in particular networked games, is to use a client-server architecture: The virtual world is managed by the server and (partially) replicated by connected clients, which visualize the scene and/or navigate an avatar through the environment. All updates from the clients are routed via the server, which can perform arbitrary filtering functions. The server is often also responsible for the simulation of autonomous entities. Timely delivery of update messages to clients is essentual to avoid visual errors (e.g. different positions of the same object on server and client). Some systems employ visibility information in order to decrease the network load, by transmitting each client only updates for those objects visible to it.

However these approaches cause a substantial overhead to the server, as it is often required to examine all objects in the environments for each client. For example, to transmit only the visible object updates to a client, it is necessary for the server to keep track of the point of view for all clients, and continously select the corresponding visible objects. Assume n = number of clients = number of objects. This means examining all objects for all clients leads to an effort of $O(n^2)$, which substantially affects the scalability. Furthermore these filtering techniques do not deal with the issue of scheduling the remaining objects; if the number of messages to be transmitted still exceeds the network bandwidth, the bottleneck problem may persist.

The technique presented in this paper aims to overcome these restrictions: it provides a *prioritized* management of the update messages from server to client, including *visibility culling* in the determinition of a message's priority. The method has constant effort $O(k)$ per client, leading to an overall effort of $O(k*n)=O(n)$ for n connected clients. Thus it is *output sensitive*, a crucial requirement for scalable environments. Our approach extends the Priority Round-Robin (PRR) queue first introduced in [Fais00].

Moreover, we introduce the concept of *activity monitoring*, an extention of PRR to cope with objects which show rapidly changing behavior and thus do not lend themselves to prioritized scheduling.

Our algorithm is directly applicable to typical current online games such as Ultima Online, Everquest, Asheron's Call, Half-Life, Quake III Arena, Unreal Tournament etc.

The remainder of this paper is organized as follows: Section 2 reviews related work, while section 3 gives a brief background of the PRR algorithm on which this work is based. It is followed by a description of the visibility algorithm given in section 4 and how it is combined with PRR. Section 5 presents an *activity*

monitoring that allows the enhanced PRR algorithm to be employed even in environments with objects having an unpredictable behavior (such as user controlled avatars). A description of the testbed and the evaluation are presented in sections 6 and 7. Conclusions are drawn in section 8.

2 Related Work

Most virtual environments employ strategies to deal with the network bandwidth restrictions that limit the number of possible update messages. Many of them are filtering techniques which reduce the number of elements competing for the resource. They can roughly be categorized into the following groups:

Several systems exploit the fact that a client is typically interested only in a small subsection – an *area of interest* - of the virtual world, which has local scope. Often the clients perception is limited to what can be seen from the current viewpoint. Objects that are occluded or too far away are not considered. Updates can be propagated on a "need to know" basis, greatly reducing the amount of messages that must be considered. The regions for which communication locality is exploited can either be given by the application designer, such as in SPLINE [Barr96], based on a regular (e.g. hexagonal) subdivision such as in NPSNET-IV [Mace95], by the viewing frustum/view cone such as in AVIARY [Snow94], or by *visibility culling* [Funk95, Makb99]. Visibility culling is often carried out with potentially visible sets (PVS) first introduced by Airey [Aire90]: An environment is first decomposed into cells, for which inter-cell visibility is pre-computed and used at runtime to identify visible objects for a given viewpoint. A simple PVS algorithm [Schm96] was also used for our test system.

A related concept is that of temporal bounding volumes (TBV) [Suda96, Suda97] and update free regions (UFR) [Makb99]. A TBV is a region of space which completely contains an object for a determined period of time. For an object in a completely hidden TBV, no update must be considered during the validity interval of the TBV. UFR implement a similar concept for mutual visibility of objects. In this paper, we construct and use TBVs to enhance message scheduling.

Communication filtering can also be performed based on proximity such as in DIVE [Benf93], or by explicitely registering interest in particular objects or events such as exemplified by NPSNET-IV or AVIARY.

A scalable distributed virtual environment requires also some care in the choice of network topology, which is often considered together with a message filtering mechanism. Several systems use multicasting instead of or together with client-server schemes to achieve better scalability. Multicast groups are often associated with a particular location or message type for implicit message filtering by multicast subscriptions. Examples for such methods can be found in NPSNET-IV, DIVE, SPLINE, and RING [Funk96]. It should be noted that although it was tested in a client-server environment, the scheduling algorithm presented in this paper is independent of network topology and can be used in any network setup.

Finally, *dead reckoning* is a networking enhancement technique used in physically based simulations where the motion of the objects is computed from linear velocity vectors. Each host stores a local copy of a remote object and predicts movements of the objects based on the current velocity. An update gets sent only when the difference between actual movement of the object at the remote host and the local copy exceeds a certain treshold. Several forms of dead reckoning have been developed, including prediction based on first-order derivatives such as in NPSNET [Mace94], position history such as in PARADISE [Sing95], or group dead reckoning such as in NetEffect [Das97].

Although all these techniques may reduce the number of messages to be transmitted by a considerable amount, they usually require a separate examination of all objects for each connected client (e.g. in visibility culling each client can have a different viewpoint), leading to a considerable effort. Furthermore, if the number of remaining messages still exceeds the network bandwidth, they must be scheduled or sorted in some way.

The scheduling algorithm presented in this paper fills in this gap. While the factors used in the algorithm are limited to a few (visual error, visibility), its freely definable metric make it principially suitable to accommodate any of the above filtering techniques, and work together with other networking techniques.

3 Background: Priority Round-Robin

The inspiration of the Priority Round-Robin (PRR) algorithm can be found in the short-term process scheduling known from operating system's research, where a set of independent processes is given processor time in order to optimize determined system's parameters [Deit90, Silb88, Stal95, Tane92]. Two of the most widely used algorithms are *Round-Robin* (or *First Come-First Served*, which is the non-preemptive version of Round-Robin), and the *Multilevel Feedback Queue* (MLFQ). Round-Robin (RR) is widely used due to its simplicity, output sensitivity and starvation-free performance, but prevents the use of priorities. The MLFQ does enforce priorities (it consists of a set of levels with decreasing priorities), but has either to deal with the risk of starvation, or must constantly monitor all processes and thus renounce to a constant overhead.

The scheduling of processes in operating systems and the scheduling of objects in virtual environments bears some substantial differences: in virtual environments for example – as opposed to process scheduling - the objects

usually need be scheduled repeatedly, and their high number prevents an examination or sorting of all objects (for more details refer [Fais00]). However, by combining the basic properties of RR and MLFQ, the PRR algorithm inherits the advantages of both, providing an output sensitive and starvation free performance, and being able to enforce priorities. It is therefore a valid replacement for RR in most circumstances. We will employ PRR in our client-server testbed to schedule position update messages, a task which is usually handled by a simple RR.

The priority management of PRR is based on the assumption that if an object is not granted the resource requested, it accumulates error, e.g. visual error. To be useful for scheduling, this error must be modeled as an appropriate error metric (such as deviation in position); the goal of the PRR algorithm is thus to *minimize the cumulative error* over all objects in the environment, called the 'overall error'.

Each object in the algorithm is assigned a so called *'Error Per Unit' (EPU)*, which is a prediction of how much the error will increase in a determined time unit. If the error is a deviation in position, then the velocity of an object is a suitable EPU.

The main loop of the PRR algorithm consists in traversing all levels simultaneously in a Round-Robin fashion, but at a varying speed which is determined by the average EPU of each level (given by the EPU of all objects contained in the level). Hence the frequency with which the objects in a level are scheduled is directly related to the (average) EPU of that level. If the levels are not traversed at a constant speed (in Figure 1 all levels have an equal speed of one), but at a varying speed, the latter can be chosen to give the objects in the levels a scheduling frequency that minimizes the average overall error accumulated by all objects.

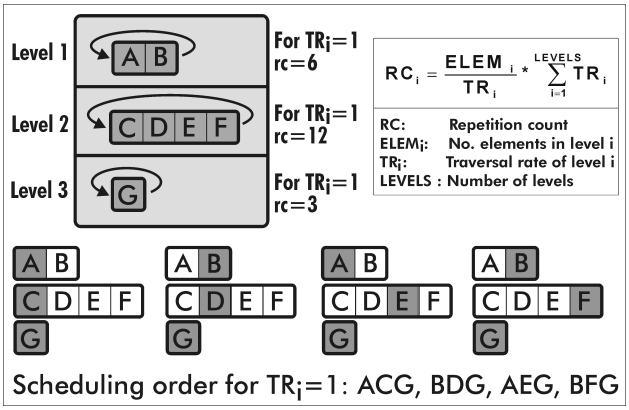

Figure 1: Scheduling order of the elements contained in the PRR algorithm if the traversal rate is set to 1 for all levels.

Let *lev* denote the number of levels and $elem_i$ the number of objects in level *i*. If we repeatedly visit all levels, each level traversed with its own 'traversal rate' tr_i, the scheduling frequency for a level (and all objects contained in it) is given by

$$rc_i = \frac{elem_i}{tr_i} * \sum_{i=1}^{lev} tr_i \qquad \text{(Eq. 1)}$$

Rc_i denotes the 'repetition count' of a level, describing the number of scheduling actions an object in that level has to wait between two subsequent selections. By minimizing the overall error *err* accumulated by all objects (we use an easy to calculate approximation)

$$err = \sum_{i=1}^{lev} elem_i * av_i * rc_i \qquad \text{(Eq. 2)}$$

the optimum traversal rate for each level is given by the number of objects in a level ($elem_i$) and their EPU (av_i is the average EPU of level *i*).

$$tr_i = \frac{\sqrt{elem_i^{\,2} * av_i}}{\sum_{k=1}^{lev} \sqrt{elem_k^{\,2} * av_k}} \qquad \text{(Eq. 3)}$$

This formula is derived by expressing the overall error with a cost function (tr_i being the variables) and solving it using the Lagrange Multipliers. A detailed description of this steps is given in [Fais00].

The main loop of the PRR algorithm consists in simultaneously traversing all levels according to their 'speed' tr_i. Every time an object is selected, it is granted the resource requested (e.g. transmitting a position update), after which the object is re-evaluated: first a new EPU is determined (we base it on the actual velocity), then the object is reassigned to one of the levels according to its EPU. Assigning the object to the level whose average EPU is most close to the EPU of the object yields an simple yet effective adaptation to even rapidly changing error distributions. Afterwards the traversal rate of the levels is modified (Equation 3) so to account for the new error distribution.

By assuming a fixed number of levels, the effort needed to schedule an object is constant; hence the PRR algorithm can achieve an output-sensitive behavior. The freely definable EPU allows us to include visibility information in the determination of an object's priority.

4 Using visibility information

4.1 Overview

Visibility information is already available in many existing virtual environments and networked games, usually employed to limit the amount of data transmitted over the network. In indoor scenes, rooms and building occlude most parts of the environments; in outdoor scenes the visibility is often limited by a radius around the user, e.g. the so called 'fog of war' in strategy games.

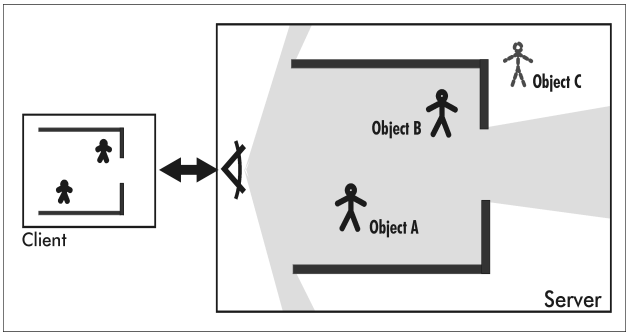

Figure 2: Visible area and visible objects for a given viewpoint of the client.

Visibility culling of objects in a virtual environment can be accomplished by first determining the visible area that can be seen from the viewpoint, and then checking which objects are inside and outside that area. Figure 2 depicts the visible area for a client, with object A and B being visible, and object C being invisible.

Usually visibility culling is first used to reduce the number of objects, then a plain FIFO or Round-Robin (RR) queue is used to schedule the remaining objects; hence the visibility information is employed to insert or remove objects from the queue.

In contrast, we replace RR with a Priority Round-Robin (PRR) scheduler and include visibility information in the priority of the objects. This allows us to reduce the effort for the server to determine which updates should be sent to each client. As each client has its own field of view, the server must usually examine all objects for each client. Assuming the number of clients approaching the number of objects, it is an effort of $O(n^2)$.

By employing the PRR algorithm it is possible to shift part of this effort the scheduler, achieving an overall effort of $O(n)$ for n connected clients. We let PRR repeatedly schedule as many objects as the network permits. Whenever an object is selected, PRR checks whether it is visible or not. For a visible object the update is transmitted, otherwise the algorithm continues its selection, looking for visible objects, with the highest speed permitted by the computing power and the network bandwidth. The visibility information affects how the object's priority is determined: visible objects get a priority equal to their velocity (their EPU); if an object is invisible, the priority is chosen such as to let the object be rescheduled when it is expected to become visible again. In our implementation we base the prediction of when an object will be visible again on the shortest path from the actual position to the next visible area (other than the actual velocity of the object).

4.2 Temporal bounding volumes

The determination of the time interval an object is supposed to remain invisible is based on a technique called 'temporal bounding volumes' (TBV). A TBV is a region of space (for simplicity often a circle or sphere) which completely contains an object for a specific period of time (called the validity interval). The TBV becomes invalid if the object leaves the volume; hence its 'expiration date' is determined by the movement of the object (e.g. rotating around a fixed point, traveling along a track, or translating freely in space) and by the size the TBV can have. In the extreme, a TBV encompassing the whole area of movement of the object will always be valid.

For objects with unconstrained translational movement, the expiration date of the TBV is directly related to its size. The validity interval of a TBV could be calculated by dividing the size of the TBV by the maximum velocity of the object. However, in large virtual environments the entities are usually avatars with an unpredictable behavior.

Our application of the TBV consists in using them to determine the priority of objects in the PRR algorithm: every time an object is scheduled, PRR determines whether it is visible or not. In the latter case, a TBV is constructed, based on the time the object is supposed to become visible again (thus, the size of the TBV determines its validity interval). Given the fact that the scheduling frequency of an object is reflected by its priority, we assign the object a priority such as to become scheduled again at the same moment the TBV expires (and the object is supposed to become visible again), providing kind of an automated 'wake-up' function. Figure 3 shows the TBV for an object with unbound translation, calculated from the shortest path to the next visible area (hatched area).

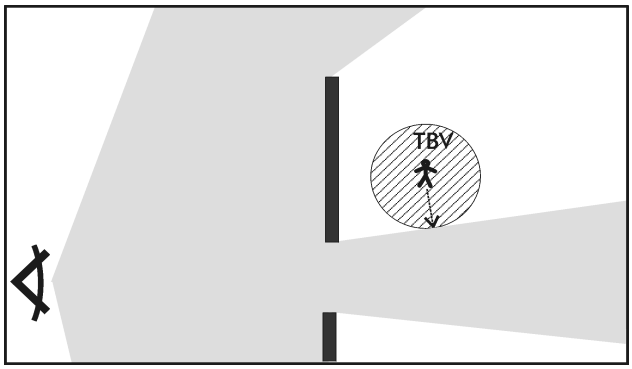

Figure 3: Temporal bounding volume for an invisible object based on the shortest path to the next visible area.

4.3 Integrating visibility information in PRR

In order to be usable by the PRR algorithm, we express the time interval an object has to wait (given by the TBV) in number of scheduling action (this value depends on the number of objects PRR can schedule per unit time). Hence we can directly compare the waiting time of an object - given by a number of scheduling actions - to the scheduling frequency of each level - given by the repetition count (as calculated using Equation 1).

An object is then assigned to that level whose scheduling frequency best matches its required waiting time.

This causes a difference in how an object is assigned to a level, depending whether it is visible or not: if an object is visible, it is assigned to that level whose average EPU best matches the EPU of the object (given by its velocity). If it is invisible, that level is chosen whose scheduling frequency best matches the waiting time determined by the TBV. In the latter case, the EPU of the object is not determined by its velocity; rather it temporarily assumes the average EPU of the assigned level. This allows the PRR algorithm to simultaneously process visible and invisible objects.

5 Activity monitoring

One possible origin of errors in the scheduling is an unpredictable or rapidly changing behavior of the objects. The Priority Round-Robin (PRR) algorithm usually computes the Error Per Unit (EPU) of an object based on its recent simulation behavior; but if the object suddenly changes its behavior by a noticeable amount, then the EPU that was computed for the object when it was last inserted into a level is no longer valid. The object would need a new EPU, but this can happen only when it is scheduled the next time.

Hence in the time interval between the change in behavior and the next scheduling of the object, the priorities and traversal rates as used by the PRR-algorithm are not correct. In the worst case, this may lead to an overall error which is worse than that produced by plain Round Robin (RR) scheduling. The scheduling frequency of the objects (given by the repetition count of the level they were inserted in) is determined by the relation of their EPUs; objects with a higher EPU get a higher scheduling frequency (a bigger share of the resource) at the expense of objects with a lower EPU. For example, an object ranked high in relation to the other objects concerning its EPU may suddenly slow down and produce an error (per unit) much lower than most other objects. But until it is rescheduled, it is bound to the fast level it was assigned to, at the expense of other objects which were previously slower, but are now faster (in relation of their EPU, alias velocity). Even worse, objects rated low and assigned to a slow level, is denied a higher scheduling frequency until they are rescheduled, in case they should experience a sudden speedup.

If such changes in behavior follow a specific pattern, the PRR algorithm can take them into consideration by analyzing the history of the object; but if the behavior is unpredictable, as occurs very often for human-controlled avatars in virtual environments, the efficiency of PRR is endangered.

Hence we have developed a measure for the 'activity' of objects, in order to quantify the frequency and 'amount' of changes in the EPU of an object (reflected from changes in its behavior). We do this by comparing the *predicted error* caused by the behavior changes (summed in a sliding average) to the *predicted benefit* achieved by using PRR compared to RR.

Every time an object is scheduled (and thus a new EPU is computed), the change between the new and the old EPU, multiplied by the repetition count between the last two schedulings, is taken as error caused by the change in behavior. This assumes the worst case, namely that the change in EPU occurred immediately after the object was assigned to the level. These errors, which include increases as well as decreases in the EPU are continuously summed up in a sliding average, called the 'error penalty'.

The predicted benefit by using PRR over RR assumes the best case, namely the difference in the overall error (between PRR and RR) that would have been experienced if all EPUs had remained unchanged; this is called the 'error benefit'. Equation 4 and 5 show the formula for the error penalty and error benefit; *noElem* is the total number of objects in the environment, epu_i and rc_i denote the EPU and the repetition count of object *i*, respectively.

$$penalty = rc_i * abs(epu_i - epuOld_i) \quad \text{(Eq. 4)}$$

$$benefit = (noElem * \sum_{}^{noElem} epu_i) - (\sum_{}^{noElem} rc_i * epu_i) \quad \text{(Eq. 5)}$$

Whenever the error penalty is higher than the error benefit for a determined amount of time (called the monitoring period), the behavior of the objects is classified as too instable to rely on priorities for the scheduling. In this case, two strategies have been tested:

- **Switching**: simply switch to RR performance, hence ignoring the priorities assigned to the objects. All levels are traversed at such a speed as to give the objects the same repetition count they would get in plain RR. This produces some undesirable peaks in the overall error when switching between PRR and RR performance.
- **Damping**: specify a maximum difference between the traversal rates of the various levels, thus limiting the influence of the priorities (EPU). We divide the interval covered by the average EPU of all levels into segments of equal length, same in number to the levels in the PRR; the length of the resulting segments is then used as maximum difference by which the average EPU of the various levels are allowed to vary. The nearer the EPU are brought together, the more the PRR approaches RR performance.

Damping is a heuristic approach, but produces good results and a smooth transition between the various stages (an evaluation is given in the section 8). Whenever the error penalty is higher than the error benefit for the monitoring period, the highest amount of damping is applied; every time that for the duration of the monitoring

period the error penalty stays below the error benefit, the stage of the difference-restriction is decreased by one.

Damping works independently of whether visibility is used in the PRR or not; it allows the PRR algorithm to become a safe scheduling strategy that can cope with almost any error distribution and objects' behavior.

6 Testbed implementation

Our testbed consists of a server which moves a determined number of objects through an environment, generated from a floor plan. A client visualizes the whole scene from a determined viewpoint and receives position updates of the various objects, in order to remain consistent with the server. Due to limited network bandwidth, only part of all pending updates can be transmitted to the client.

The visual error, given by the difference in position of the objects on server and client, will be minimized by the enhanced PRR algorithm. While the simulator continuously moves all objects, we let the PRR algorithm select as many objects as the network permits (e.g. 10 %), using the velocity of the objects as Error Per Unit (EPU). As reference we use the same setup, but with a plain Round-Robin (RR) queue instead of the PRR algorithm. Like PRR, the plain RR transmits only the visible objects to the client, but does not enforce priorities.

Figure 4 shows a snapshot of the evaluation testbed: the server (simulator), located in the top panel, depicts the floor plan and the moving objects; the client's viewpoint is depicted by a star, and the invisible areas are shown shaded dark. The lower two panels show a visualization of the visual error of the connected client; the third panel (from the top) depicts the visual error for RR, while the bottom panel shows the visual error for the enhanced PRR. The visual error is depicted by a line from the actual position of an object (on the server) to the last updated position (on the client) - thus the longer the line, the higher is the visual error. The graph in the panel underneath the simulator permits to monitor the overall error for RR and PRR scheduling.

The motion of the objects through the environment was implemented by first digitizing and triangulating a floor plan, and then generating a connection-graph of the triangles. The simulator generates for each object a path from the current position to a random destination position, and then moves the object along this path with a given velocity (used as EPU).

The area visible from the viewpoint chosen by the client is also easily computed with the help of the triangulated floor plan. Starting from the triangle which contains the viewpoint, the 2.5D-visibility algorithm presented by Schmalstieg in [Schm96] generates the set of potentially visible triangles from the viewpoint.

If the object is invisible, then the algorithm determines the shortest path from the actual to the nearest visible triangle; from the length of the path and the actual velocity PRR makes a safe guess of the moment the object will become visible in the worst case (if it immediately starts heading for the visible area), and gives the object an according priority.

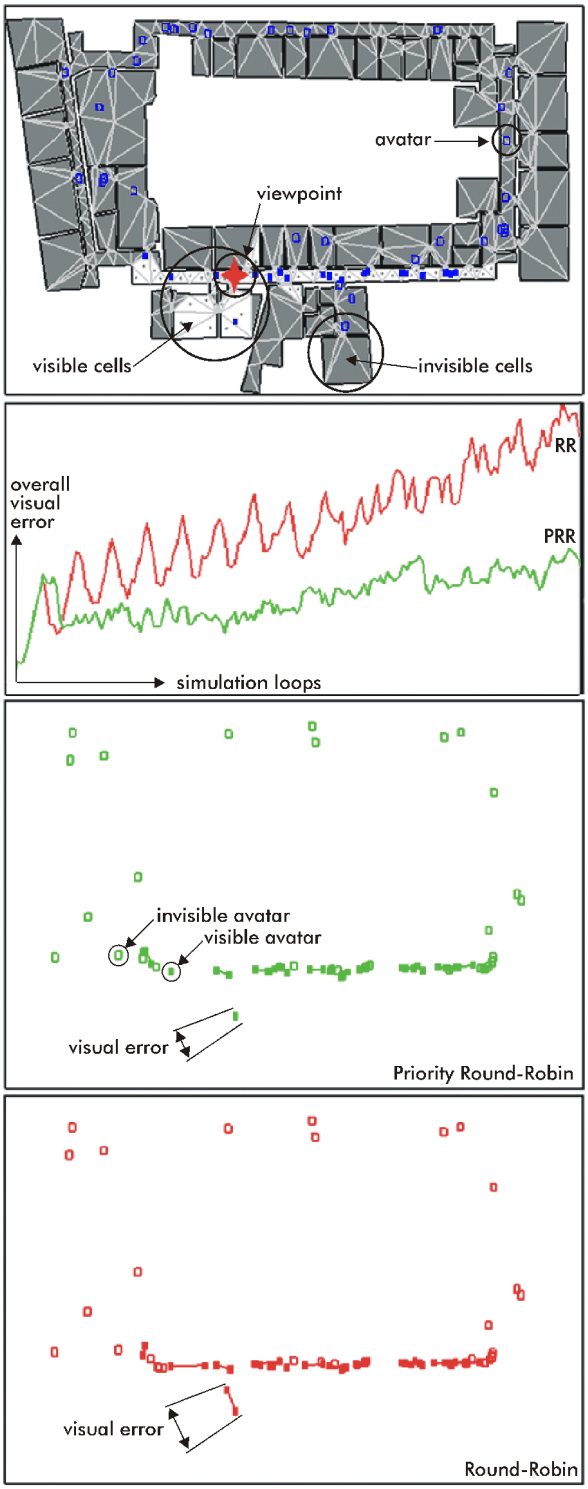

Figure 4: Screenshots of the testbed employed to evaluate the enhanced Priority Round-Robin algorithm.

7 Evaluation and results

The enhanced PRR algorithm is evaluated by comparing it to plain Round-Robin (RR) scheduling; this allows us to evaluate the performance increase that can be gained by substituting RR with PRR.

In the example given below, the server hosts a simulator and moves 10000 objects (simulating avatars) at a predetermined velocity through the environment; the velocity is used as Error Per Unit (EPU). To simulate the network bottleneck, although the simulator can move all objects in every simulation step, only 10% of the position updates (1000 in numbers) can be transmitted to the client. The main loop of the testbed consists thus in first simulating all 10000 objects; afterwards a PRR-scheduler, as well as a plain RR queue can select and update 1000 objects. The actual overall error is computed and evaluated for both RR and PRR scheduling after each loop.

7.1 Example 1: clustered error distribution

This example shows a case apt for the PRR algorithm, as we have three different clusters of EPUs which can be serviced by PRR at different priorities. We let 10000 avatars walk along random paths in the environment at different velocities (used as EPU):

- 1000 avatars get a velocity between 2.9 and 3 units
- 2000 avatars get a velocity between 0.5 and 0.6 units
- 7000 avatars get a velocity between 0.1 and 0.2 units

The camera is placed in the same location of the map as can be seen in Figure 4. Figure 5 shows a comparison of overall visual error caused by a RR and a PRR algorithm (for the same client), if only 10% of the 10000 objects are scheduled after each simulation loop.

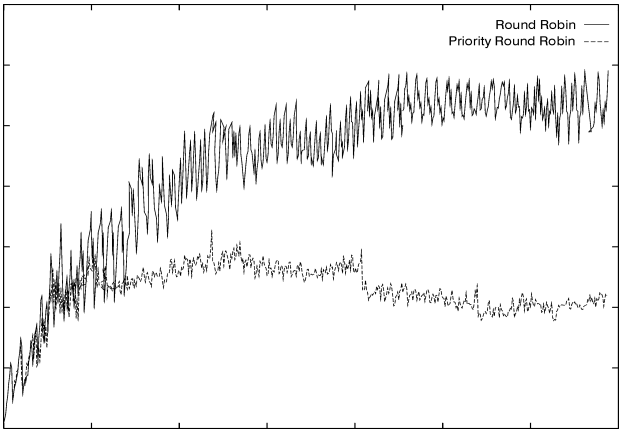

Figure 5: Due to the clustered error distribution, the visual error of the enhanced PRR is 130.7% lower compared to RR.

7.2 Example 2: uniform error distribution

As PRR relies on servicing the objects at different priorities, according to their EPU (velocity), a uniform error distribution prevents PRR from constructing clearly distinct error groups. In contrast to the previous example, each avatar gets a random velocity between 0.1 and 3 units. Hence the reduction of the overall visual error, as compared to RR scheduling, is 'only' 85.1%.

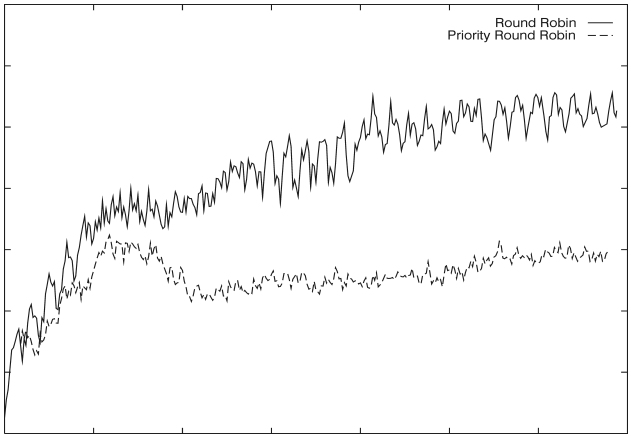

Figure 6: With a uniform error distribution, the visual error of the enhanced PRR is 85.1% lower compared to RR.

7.3 Example 3: unpredictable movement

In this example we produce a situation that may be critical for PRR, but is likely to happen in virtual environments, and especially networked games where the motion of objects is directly controlled by a user. If the objects experience a fast-changing, unpredictable behavior, it makes a correct prediction of the behavior impossible. Hence the PRR algorithm is not able to correctly enforce priorities, as an object can experience a sharp change in velocity (and hence of its EPU, its contribution to the overall visual error) even immediately after it has been assigned to a given level according to its behavior prediction.

To simulate this situation we give all 10000 objects a random velocity between 1 and 10 units; but after 10 simulation loops we shuffle the velocities of all objects. An extreme behavior to this extent will rarely happen, but it clearly shows the efficiency of the failure-safe mechanisms in the enhanced PRR algorithm: a heuristic is used to detect whether the 'benefit' gained by using PRR over RR exceeds the deterioration caused by an erroneous prediction of the objects' EPU. In such a case, the PRR temporarily reduces the enforcement of priorities, as described in Section 5.

Figure 7a shows a comparison of the overall visual error produced by PRR and RR, if the 'damping' mechanism is activated; in this case PRR makes the best out of such an extreme situation (*all* objects perpetually change behavior), and produces a performance close to RR (better only by less than 1%). If a PRR, with no safety mechanism, is employed (see Figure 7b), the erroneous prediction of the priorities lead to an overall visual error even worse than RR by 9.1%. Fortunately the employed heuristic can provide security at no discernible expense.

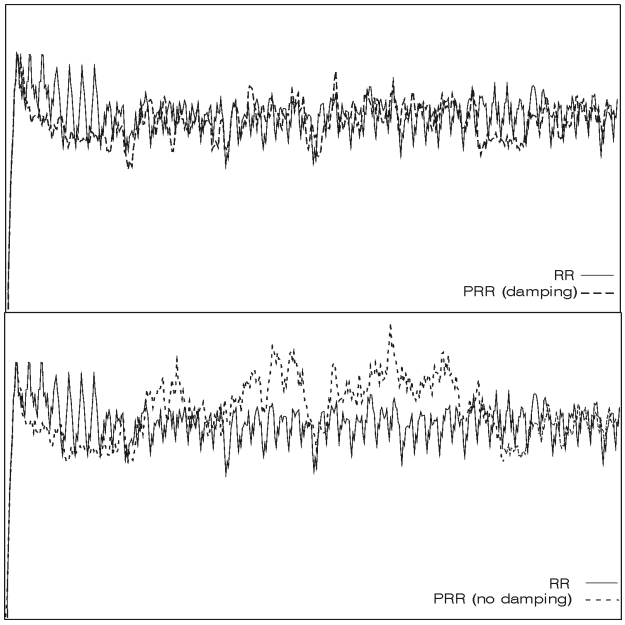

Figure 7a, 7b: Performance of Priority Round-Robin w/o damping if the objects have an unpredictable behavior.

8 Conclusion and future work

The enhanced Priority Round-Robin (PRR) algorithm presented in this paper can bring a substantial contribution to the development of distributed virtual environments or networked online-games which contain a very high number of objects. It allows to handle the transmission of update messages from server to client at a constant effort per connected client, determining the frequency of the updates from priorites based on the behaviour of the objects as well as visibility information. It can substitute the plain Round-Robin queue used in virtual environments to transmit the update messages to the clients at no discernable effort, providing a scalable technique that leads to a graceful degradation of the systems performance caused by network bandwidth limitations. Our experiments confirm that due to its activity monitoring, the enhanced PRR algorithm is superior to conventional scheduling in real-world situations even if objects may have unpredictable behaviour, such as user-controlled avatars.

In order to further optimize the enhanced PRR algorithm to be employed in distributed environments and online-games, future work will examine the scheduling of avatars which are build according to a hierarchical human model, and employ Levels of Detail for the accuracy of the updates transmitted to the clients. It is planned to construct an extended environment containing a large number of rooms, buildings and open landscapes, and evaluate perceptual error metrics to minimize the visual error as perceived by the user.

Acknowledgements

The work presented in this paper was sponsored by the European Community under contract no. FMRX-CT-96-0036.

References

[Aire90] J. M. Airey, J. H. Rohlf, F. Brooks Jr.: Towards Image Realism with Interactive Update Rates in Complex Virtual Building Enviroments. Computer Graphics, 24(2):41, 1990.

[Barr96] Barrus, J., Waters, R., & Anderson, R.: Locales and Beacons: Precise and Efficient Support for Large Multi-User Virtual Environments. Proceedings of VRAIS'96, pp. 204-213, Santa Clara CA, 1996.

[Benf93] S. Benford, L. Fahlen: A spatial model of interaction in large-scale virtual environments. 3rd European Conference on CSCW, pp. 109-124, 1993.

[Das97] T. Das, G. Singh, A. Mitchell, P. Kumar, K. McGhee: NetEffect: A Network Architecture for Large-scale Multiuser Virtual World. Proc. of ACM VRST'97, pp. 157-163, 1997.

[Deit90] H. M. Deitel. An introduction to operating systems. Addison-Wesley, Inc. ISBN 0-201-18038-3, 1990.

[Fais00] C. Faisstnauer, D. Schmalstieg, W. Purgathofer. Priority Round-Robin Scheduling for Very Large Virtual Enviroments. Proc. of IEEE VR'2000, pp. 135-142, 2000.

[Funk95] T. A. Funkhouser. RING - A Client-Server System for Multi-User Virtual Environments. SIGGRAPH Symposium on Interactive 3D Graphics, pp. 85-92, 1995.

[Funk96] T. Funkhouser: Network Topologies for Scaleable Multi-User Virtual Environments. Proceedings of VRAIS'96, pp. 222-229, Santa Clara CA, 1996.

[Mace94]M. R. Macedonia et al. NPSNET: A Network Software Architecture for Large-Scale Virtual Environments. Presence, Vol 3(4), pp. 265-287, 1994.

[Mace95]M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, P. Barham: Exploiting Reality with Multicast Groups. IEEE Computer Graphics and Applications 15(3), pp. 38-45, 1995.

[Makb99] Y. Makbily, C. Gotsman, R. Bar-Yehuda: Geometric Algorithms for Message Filtering in Decentralized Virtual Environments. SIGGRAPH 1999 Symposium on Interactive 3D Graphics, pp. 39-46, Altanta GA, April 1999.

[Schm96] D. Schmalstieg et al.: Demand-Driven Geometry Transmission for Distributed Virtual Environments. Proceedings EUROGRAPHICS '96, 15(3), 421-433.

[Silb88] A. Silberschatz. Operating system concepts. Published by Addison-Wesley, Inc. ISBN 0-201-18760-4, 1988.

[Sing95] S. K. Singhal, D. R. Cheriton. Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality. Presence, Vol. 4 (2), pp. 169-193, 1995.

[Snow94] D. N. Snowdon, A. J. West. AVIARY: Design Issues for Future Large-Scale Virtual Environments. Presence, Vol 3(4), pp. 288-308, 1994.

[Stal95] W. Stallings. Operating systems. Published by Prentice-Hall, Inc. ISBN 0-02-415493-8, 1995.

[Suda96] O. Sudarsky, C. Gotsman. Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. Proceedings of EUROGRAPHICS'96, Vol 15(3), pp. 249-258, 1996.

[Suda97] O. Sudarsky , C.Gotsman: Output-Sensitive Rendering and Communication in Dynamic Virtual Environments. Proc. of ACM VRST'97, Switzerland, 1997.

[Tane92] A. S. Tanenbaum. Modern operating systems. Published by Prentice-Hall, Inc. ISBN 0-13-588187-0, 1992.