Dynamic Load Balancing in Distributed Virtual Environments

Stephan Mantler and Dieter Schmalstieg

Vienna University of Technology, Austria
{step | dieter}@cg.tuwien.ac.at

Abstract

This paper introduces a new approach for improving the scalability of distributed virtual environments by using a combination of visibility culling for communication and dynamic load balancing to keep the system load evenly distributed. Further communication optimizations as well as some preliminary results are presented.

1. Introduction

In many Virtual Environments, there is a desire for huge numbers of simultaneous participants within the environment. However, these goals are defeated by limits of current software and hardware. Increasing load will degrade responsiveness to the point where the system is rendered useless. To overcome the restrictions, distributed designs are needed that improve the *scalability* of the system.

An optimal design will therefore minimize the communication through a careful decomposition of the system; additional computational effort can be used to further reduce the required communication, however care must be taken that the increased processing requirements will not outweigh the improvements of distribution.

An efficient networking mechanism for that purpose is the use of multicast communication [1]. In addition to using a generic distributed approach, improvements can be obtained by exploiting properties specific to the virtual environment application. For example, if the environment features dense occlusion, such as in building interiors, the additional visibility information can be used to reduce the number of messages sent to each client. This reduction is typically performed by a network of servers, each of which possesses geometry (and thus visibility) information about some part of the environment as well as the clients in this part. As a client can see only a small fraction of the total number of avatars in such a system, filtering based upon visibility information greatly reduces the number of messages that need to be passed, which leads to a significant increase of the system's scalability.

Funkhouser described an architecture named RING [2] which uses static visibility information to perform message filtering at runtime. The environment is decomposed into adjacent parts, which are statically distributed over a network of servers.

More specifically, the system's scalability is now not longer limited by the total number of clients in the environment, but the number of clients that need to be accounted for by one server. This is without doubt a drastic improvement, however it is still possible for such a system to become overloaded as soon as one server reaches its limits, even if the other computers are only very lightly loaded.

This paper introduces an architecture which is aimed at taking the scalability of distributed virtual environments one step further by employing *dynamic load balancing* to keep all servers evenly loaded. The virtual world is decomposed into regions, the distribution (and hence the cost of maintenance) of which is balanced among the available servers. This architecture differs from previous design by allowing a dynamic reconfiguration of the simulation subdivision among servers, including database and network connections. Our approach also allows to add and remove servers at runtime, which is essential for Internet-based applications.

2. A system design with load balancing

In order to keep the system's load distributed evenly over all connected machines, some sort of workload balancing needs to take place. In the above case of assigning a part of the environment to each server, an overloaded server will aim to reduce its load by reducing the part of the world it has to handle, while a server noticing that it has little or nothing to do can request further work. In this section, we outline our system design, which was inspired by the RING

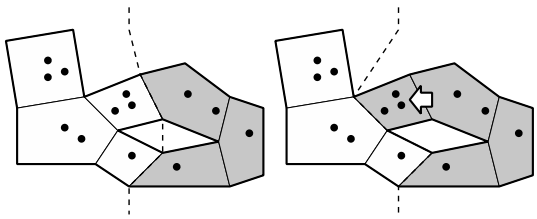

Figure 1: *Regions are transferred between servers to balance system load.*

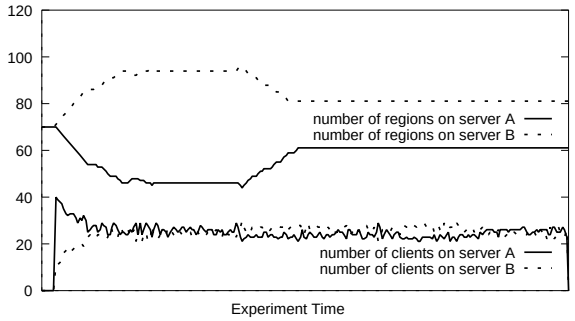

Figure 2: *Load balancing in action: Initially server A has too many clients and gradually passes regions and clients to server B. As the clients move about randomly, they become evenly distributed over the world and cause B to return some regions to A.*

architecture, but is capable to include load balancing among servers.

For simplicity, we have limited the environment to a $2\frac{1}{2}$D representation, which is sufficient for the representation of building interiors. Visibility can be determined from the floor "map" with a fast on-the-fly algorithm [3], which allows rapid point-to-cell and cell-to-cell visibility computation. The world consists of a number of adjacent, non-intersecting convex polygons, where each edge of the polygon can be either transparent - representing an open door - or opaque for walls. The size of these polygons determines the granularity of the load balancing.

The simple heuristic that has been used in our implementation tries to estimate the server's load by examining the number of clients as well as the regions. By trying to keep the client count below a certain level, the network traffic and required computing power can be limited; the tendency towards keeping the number of regions down tries to keep an otherwise unloaded server from "bad surprises" like a large number of clients entering its regions.

If it is decided that load balancing is required, the heuristic tries to choose a region for transfer that will bring the most benefit for the server, i.e. the one that will move as many clients as possible to the other server.

Naturally, such transfers should not cause another server to become overloaded and necessitate additional region transfers, since this could cause unwanted oscillations as regions and clients are handed over back and forth. Therefore, there are two limits on the load scale:

- The *high water mark* defines the point above which a server tries to get rid of a region.
- If a server's load falls below the *low water mark* it will try to take over a region from another server.

The latter is needed to evenly distribute the load even in areas where the overall load is low by requiring "bored" servers to take over some of the work from other servers.

3. Evaluation and Results

We have implemented a prototype of the design as described in the previous sections and are currently performing experiments to examine the reaction of the environment to the change of parameters (world size, number of servers, number of clients, directed client movement vs. random walk). To simulate a large number of human participants and to be able to reproduce the test situation under changing conditions, the movement of the clients is controlled by "robot" programs which are able to explore the world without user interaction. Preliminary results show that our approach is feasible and appear encouraging.

To conclude, the presented system can be regarded as a framework for further studies. This future work will be aimed at examining potential heuristics and performance optimizations as well as the integration of the system with other ongoing research.

Acknowledgments This project is sponsored by the Austrian Science Foundation (*FWF*) under contract number P11392-MAT.

References

1. M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, and P. Barham, "Exploiting reality with multicast groups: A network architecture for large-scale virtual environments", *Proc. VRAIS'95*, (1995).
2. T. Funkhouser, "Network topologies for scalable multi-user virtual environments", *Proceedings of VRAIS'96, Santa Clara CA*, pp. 222–229 (1996).
3. D. Schmalstieg and R. Tobler, "Exploiting coherence in 2.5 d visibility computation", *Computers and Graphics*, **21**(1), pp. 121–123 (1997).