Distributed Rendering in Heterogenous Display Environments - A Functional Framework Design and Performance Assessment

Seipel S. and Ahrenberg L.,
Department of Information Technology, Uppsala University

ABSTRACT

With this paper we focus on complex display environments in which several users view upon numerous types of 3D displays. We present a method for synchronization and shared state management for independent rendering processes. Our approach is based on TCP/IP based virtual shared memory architecture and intelligent clients in order to accomplish state coherent rendering on multiple displays. We describe a series of benchmark tests that we used to identify frame-to-frame incoherency for rendering of animated objects. The results of these tests allow for a quantitative assessment of the underlying distribution model for networked rendering.

Keywords

Distributed Rendering, NetVR, Display Environments

1. INTRODUCTION

In many applications from industrial design to process control, retrieval of complex information and its appropriate visualization has become a group work task that must be accomplished in a collaborative manner. In the past, different post-desktop computer interfaces have been investigated, which allow for collaborative work in an environment where participants are co-located in the same physical space. Typical examples are the iSpace project at Stanford University [1]. Common to these approaches is that they are based on conventional i.e. 2D human-computer interfaces.

At the Swedish Defense College the potential of collaborative virtual environments has been recognized and research has been initiated (project AQUA) that investigates advanced 3D visualization techniques for command and control [2]. The current configuration of the AQUA visual environment consists of one horizontal large screen display, and four stereoscopic large screen retro-projectors, which are arranged in the corners of the AQUA environment (see figure 1). These displays are viewed by up to 10 users that communicate with one another in the AQUA environment. In addition, each user has at least on local computer display for individual non-collaborative work.

In the AQUA project a general assumption is made that all displays are 3D displays, and that there are arbitrary number and arbitrary spatial orientations of the displays in the physical environment. In consequence all information to be visualized is represented in a thought virtual space that is metrically aligned with the physical space and all displays that are presented in the physical AQUA environment are defined by their corresponding windows-on-world parameterization in the virtual space.

Since the AQUA environment is supposed to be a general 3D visualization environment there is a large number of individual viewing parameter configurations depending

on which user is watching on what display at a certain given point in time.
Rendering of 3D graphics in this multiple-display environment is naturally accomplished by using clusters of independent hosts. Hence, all information that is simultaneously visualized on different screens must be distributed and shared among the involved rendering engines.

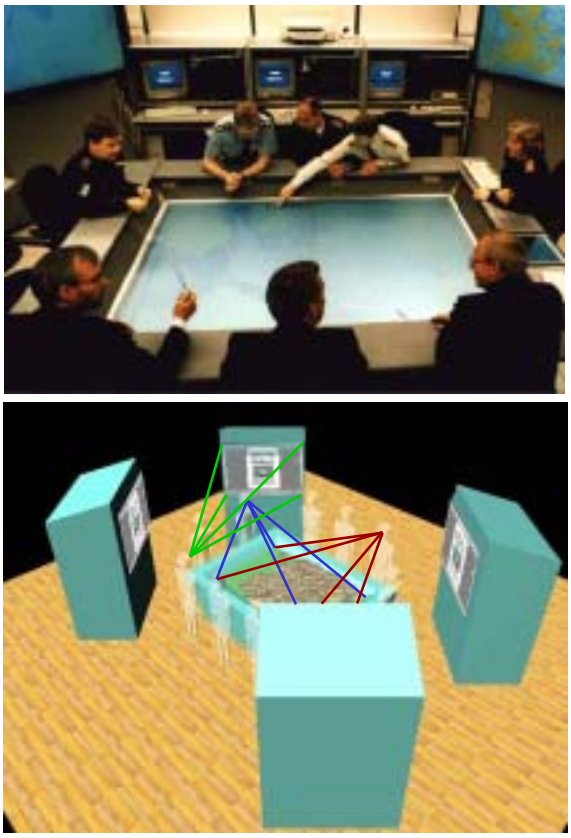

Figure 1. View upon the physical environment of the AQUARIUM (above). For 3D representations various individual viewing frustums must be maintained (below).

2. METHOD

In this project we developed a shared state database for accomplishing simultaneous and distributed 3D rendering of remotely shared virtual objects. It also provides means for runtime re-configuration of the viewing parameters for all involved rendering clients in a complex visualization scenario.
The primary goal of this shared state database is to maintain good runtime performance for a limited number of rendering clients (up to 30) and for a relatively low number of shared states (up to 10.000 floating point values). We also anticipate that rendering applications are executed concurrently in the local area network of a local PC rendering cluster. Unlike other architectures for building distributed VR environments as e.g. DIVE [3], it is not the intention to provide a generalized and fully replicated scene-graph database. We also wanted to avoid a very specialized low level implementation of a specific simulation protocol as e.g. found in the SIMNET environment [4]. Instead, our conceptual approach builds upon intelligent clients that administrate their individual scene-graphs independently. They are using minimal state change propagation to maintain consistency.

On the network level we implemented transparently usable shared memory architecture - STREEP [5]. This library is based on a TCP/IP based protocol and it facilitates allocation of virtual memory, propagation mechanisms for state change updates, and subscriptions for process notification.

Based on this virtual shared memory architecture, applications in the AQUA environment can share relevant information in so-called “pools”. Figure 2 illustrates our concept of shared pools. A pool can be considered as a shared memory area that can be allocated by 3D clients or to which clients can subscribe. A pool contains data of the same type. An example of a pool is a projector pool that contains the parameter configuration to defining the projection pipeline for an individual user looking at a specific display. A pipe pool contains information about the number and configuration of visual channels on a specific display. Other pools are e.g. sensor pools, which store information from various tracking devices, and shared data pool that contains shared data, which is actually to be visualized in the AQUA environment.

Apart from assuring state consistency, the minimization of network latency is one of the most critical issues in distributed rendering. This is in particular true for applications, where the result of the 3D rendering process is visualized simultaneously in the same physical environment. Here, the synchronization of graphical states (i.e. transformation matrices) in the local rendering processes must ideally be frame consistent. In other words, animated objects and other animated states (illumination etc.) must be rendered in the same attitude on all displays at the very moment. Absolute frame consistency cannot be guaranteed with a purely software based synchronization method based on virtual shared memory as in our proposed framework. On the other hand we can ague that absolute frame consistency might not be necessary since the human visual apparatus has limitations both in regard to spatial and temporal resolution.

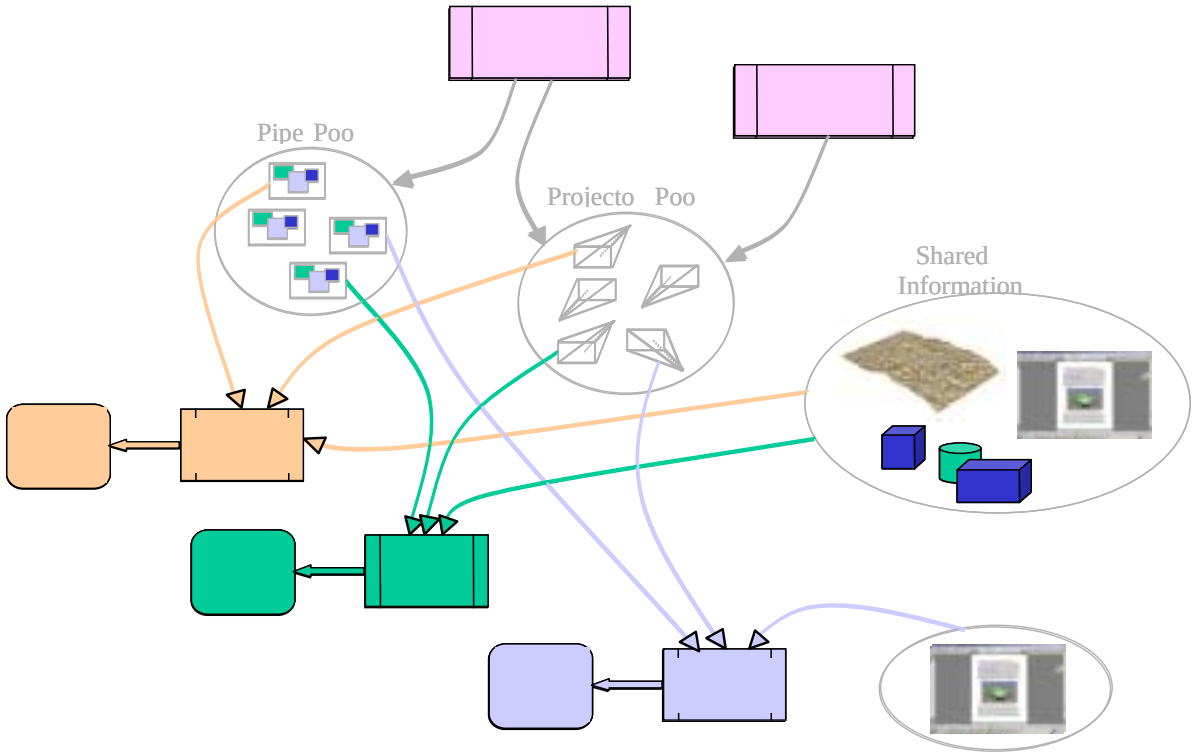

Figure 2. Illustration of the shared data pool concept. Different applications share data for rendering purposes, which is shared a network based shared memory.

In order to shed more light upon this issue we performed an initial runtime performance study. Its goal is to measure and quantify visual artifacts as a consequence of delayed state propagation in our concept for distributed rendering.
To that end, we designed a generic server application that manipulates the states of one (or many) shared objects. Two client applications were designed that render the object with exactly the same viewing conditions. The output of those two independently running client processes is rendered into equally large view ports on the same host computer. Hence, in the ideal case of absolute frame consistency, the graphical output of both processes should be identical for moving objects. In the practical case however, we expect differences in the visual output of these processes as described above. This basic program set-up was used to benchmark the real-time performance under different conditions.

Test set-up:
Two client applications where executed on the same computer, and each was reserved half the available screen space. Both processes where rendering a shared object that was in continuous motion (see figure 3). The test object is a very simple fan geometry rotating with a predefined angular velocity of 360 degrees per second. The actual rotational angle was not manipulated by the client applications, but by a third server process, that also maintains the shared state (rotation) for this fan.

Assessment method and criteria
The client applications where run simultaneously on one computer and a software based frame grabber application (Camtasia by TechSmith) was used to capture the entire screen content i.e. both application windows at a certain frame grabbing rate (20Hz). A sequence of 200 frames was recorded in this manner. In order to assess frame and state delays, we developed a program that superimposes the graphical output of both client applications, and that counts the number off different pixels in those pictures. For absolute frame consistency, the output should be identical and hence the number of differing pixels should be zero. An increasing number of differing pixels indicates frame delay. Since the number of pixels increases in big intervals, every interval indicates the delay of exactly on frame.

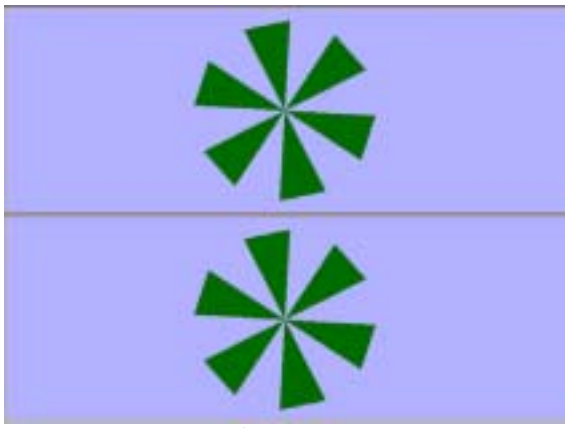

Figure 3. Screen-shot of a test setup. Two independent applications render an animated object, whose rotational state is shared through network based shared memory.

3. RESULTS

Based on the general set-up, we performed three tests under different conditions. Test A: 2 Clients windows are opened on the same computer, the screen resolution is chosen to be 800x600 pixels to allow for higher screen capturing rates. The server is running on a separate remote computer, which resides in the same sub-network. The observed variable is the difference in pixels in the two client windows over time i.e. the number of frames that are out-of-sync. This experiment is repeated several times, whereby the rate of the object updates from the server side is increased. Observe, that an increased update rate on the server side implies relatively small angular increments because the angular velocity was chosen to be constant with 360degrees/sec. The goal is to study the relation between frame delay and shared state update rate.

Table 1 shows the data measured for 200 animation frames. They where measured by capturing at different object update rates. The observed variable is the number of differing pixels in the two client windows, and the data was sorted in descending order. For clarity, only the first 20 data sets are shown in the graphics, because the remaining values are zero altogether. At lower update rates, the angular increment per frame is higher given a constant angular velocity. This explains that in table 1, the difference in the pictures at 10Hz object update rate is higher than compared to the 40 Hz or 50 Hz situations, where there is less angle increment per animation frame.
The figures show that for 10 Hz object update rate there is only 1 out of 200 frames delayed. At 20 Hz and 30 Hz object update rate, there are 7 or 6 frames out of 200 frames delayed by one step. The number naturally increases as the object state update increases, but it does not exceed 11 frames delayed per 200 animated frames.

Interesting to observe is, that even for higher object update rates the number of delayed frames in the client windows does not exceed one frame.

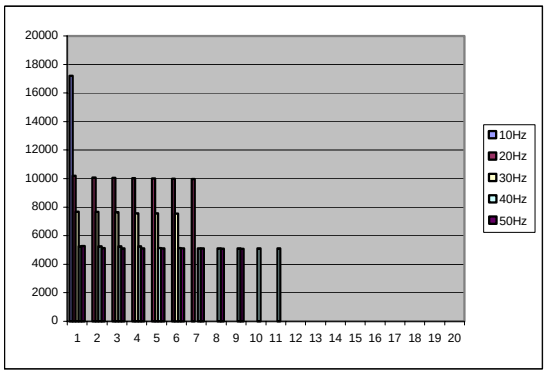

Table 1. The number of differing pixels in between the two client frames sorted by magnitude in descending order for different state update frequencies.

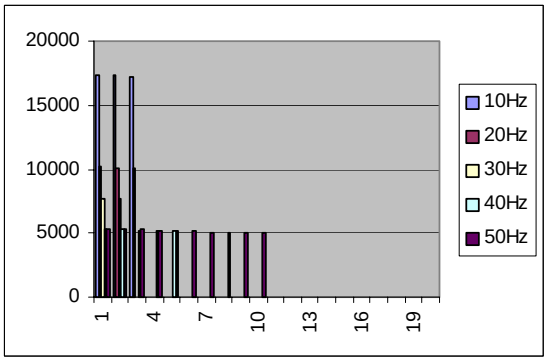

Table 2: The number of differing pixels in between the two client frames sorted by magnitude in descending order for different state update frequencies (server and visualisation clients on same host).

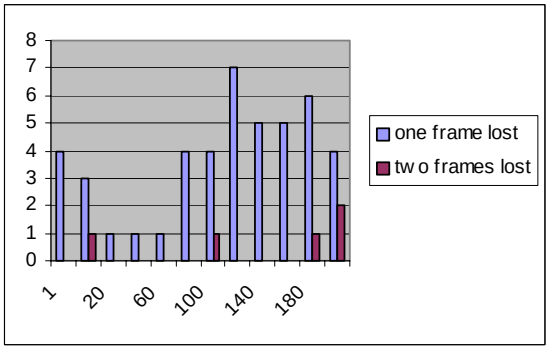

Table 3: The number of in-coherent frames out of 200 frames rendered for increasing number of shared states.

Test B: The test was repeated with the same conditions as in Test A except for the fact that now the server (the process manipulating the object state) is also running on the same host as the two client processes. We would expect that in this condition network routing efforts be reduced, since all TCP/IP traffic is routed on the local host rather than being passed through the Ethernet. Table 2 shows the result for this test. The observed values show a similar pattern as in test A i.e. for increasing state update rates, the number of frames out of sync is increasing. For almost all state update rates, the number of dropped frames was less than in test A. Remarkable is that for the lowest update rate of 10Hz a one-frame delay could be observed three times.

Test C: Another test was carried out to study how much the total network traffic affects frame consistency. In the previous tests, the server only had to maintain the shared attributes of one single object (one fan). In the following experiment the set-up was as in Test A: Two clients windows opened on the same host, the screen resolution set to 800x600 pixels. One is server running on a remote host. The object animation update rate was set to only 10Hz server side. Variable parameters in this test set-up were the number of distributed objects, which was increased stepwise from 1 object to 200 objects. The result of the observations is visualized in table 3.

The result of this experiment shows that up to a number of 100 shared objects there are only between 1 and 4 frames incoherent in an animation sequence of 200 frames. Further increase of the total network traffic (i.e. number of shared objects) will also increase the number of incoherent frames up to seven frame-mismatches at 180 updated objects. Worth mentioning is the fact that with an increasing number of objects that are shared, the frame delay between the two rendering processes is not only one frame, instead the frame incoherence starts to stretch over two animation steps.

4. DISCUSSION

Prior to interpretation of the results the methods of testing must be critically discussed. This initial benchmarking was performed by adopting a very straightforward and easily to implement measuring method using a frame grabber software. This means that the processor load induced by the frame grabbing application introduces artefacts, since less processing power is available to the rendering applications. On the other hand, this speaks rather for an optimistic interpretation of the measured results, because the frame grabber application could indeed be a reason for observed frame delay, which would not occur without running the grabbing process. Due to the need for simultaneously capturing the rendering result of two processes, both rendering clients were running on the same single processor computer system. Load balancing is managed by the operative system and is therefore an uncontrollable factor in the test. Asymmetric task scheduling might therefore be an additional cause for frame inconsistency. After all, the characteristics of the test set-up suggest that we actually should expect better network throughput performance and therefore we do not see a reason to mistrust or pessimistically interpret our results.

5. CONCLUSION AND FUTURE WORK

Summarizing our first observations we can conclude that the TCP/IP based propagation of shared states performs surprisingly well. In a local area network we can expect that about 5 frames in a sequence of 200 animation steps will be out of synchronisation for an average object update rate of 30 Hz (which appears sufficient for almost all object animations in a 3D scene). These frame incoherencies mean only delay of a single animation step. It depends actually on the speed of simulated objects and the scale how dominant this one-frame incoherence is perceived by the user. In regard to the total network traffic we can see that for 200 shared states the proportion of dropped frames is about 10 out of a total of 200 animated frames i.e. 5% of all animated frames are not synchronised. At this stage we can conclude that the runtime performance exceeds our expectations and is more than satisfactory for applications, where a limited number of shared states (below 1000) needs to be synchronized.
It remains to be seen in our future studies, if and how these figures will improve when rendering processes are running exclusively on dedicated hosts. The will also show how performance will scale for increasing numbers of shared states. Our future studies will therefore incorporate superimposition of multiple independent computer displays that will be captured using high-speed video cameras rather than software-based capturing of several application windows that are executed concurrently on one single host. Another issue that needs to be addressed is the question to what extend the human user is capable of perceiving dropped frames or frames that are not fully synchronized. In order to explore that, further user-oriented tests will be carried out to measure visual-perceptual artefacts in different situations of state incoherency.

REFERENCES

[1] Fox, Armando, Brad Johanson, Pat Hanrahan, and Terry Winograd, Integrating Information Appliances into an Interactive Space, *IEEE Computer Graphics and Applications* 20:3 (May/June, 2000), 54-65.

[2] Sundin C. and Friman Henrik (eds.) ROLF 2010 – The Way Ahead and The First Step, Försvarshögskolans Acta C6, Elanders Gotab, Stockholm 2000

[3] Carlsson, C. and Hagsand, O. DIVE – a Multi-User Virtual Reality System. *IEEE VRAIS*, Sept 1993.

[4] Pope, A. The SIMNET network and protocols.*Technical Report 7102.* Cambridge, MA: BBN Systems and Technologies, July 1989.

[5] Lindkvist M: A state sharing toolkit for interactive applications. *Master Thesis.* Department of Information Technology, Uppsala University. 2001.