Client-Server Infrastructure for Interactive 3D Multi-User Environments

Ingo Soetebier
Fraunhofer IGD
Fraunhoferstr.5
64283 Darmstadt, Germany
ingo.soetebier@igd.fhg.de

Horst Birthelmer
Fraunhofer IGD
Fraunhoferstr.5
64283 Darmstadt

Jörg Sahm
Fraunhofer IGD
Fraunhoferstr.5
64283 Darmstadt
joerg.sahm@igd.fhg.de

ABSTRACT

Multi-user environments, using 3D graphics, more and more find their way into areas like e-business, entertainment and cooperative work. Due to the increasing capabilities of hardware and network, even smaller devices, like laptop computers or PDAs, become suitable for the visualization of 3D graphics. A possible approach for distributing the 3D information is to use a server, which provides all clients with the necessary data. In this paper we define a client-server software framework, which supports the design of three-dimensional multi-user environments. We define requirements of the software infrastructure for a client-server framework that can provide a number of different 3D scenes, each with several users. The number of 3D scenes and users should be only limited by the hardware. Furthermore, we present a solution for each requirement and finally we discuss the results.

1. INTRODUCTION

Since the capabilities of 3D hardware and network capacity increase in very short cycles while the prices are dropping rapidly, there's a big demand for interactive 3D environments. Areas for 3D environments are engineering and design where users working on the same projects can review and discuss results in a 3D environment.

To support the above-mentioned scenarios, a powerful software framework for building the client-server environment is necessary.

1.1 Related Work

Three-dimensional multi-user environments heavily depend on network connections. There are papers from Funkhouser [Fun96b] and Macedonia et al [Mac97a], which focus on the classification of network connections and network topology.

There are some multi-user environments, which are based on multicast peer-to-peer communication. *NPSNET* [Mac94b] is an example for such an virtual environment. It is designed for a large number of users in military training and is based on the DIS protocol that uses multicast IP. There is no central server in *NPSNET*. *Bamboo* [Wat98b] was developed as an advancement to *NPSNET*. It supports a dynamic extensibility at runtime. In *Bamboo* this feature is used to supply network protocols per object to achieve optimal performance of transmission. A virtual reality system using also multicast peer-to-peer communication is the *Distributed Interactive Virtual Environment (DIVE)* [Car93a]. Environments used by *DIVE* are stored in databases, which are replicated on each client. The *Scalable Platform for Large Interactive Network Environments (Spline)* [Wat96a] is a toolkit for creating large-scale multi-user environments. Its world model is split up into several so-called 'locales'. Communication is done via multicast peer-to-peer connections. A remarkable application built with *Spline* is DiamondPark.

Like our approach, there are some multi-user environments, which are based on a client-server communication. An example is the *AVIARY VR system* [Sno94a] presented by Snowdon et al. *Virtual Society* [Lea97a] presented by Lea et al is another client-server based virtual environment. It is an application that uses 3D worlds defined with VRML and an employed communication protocol called Virtual Society Client Protocol (VSCP). The *MR ToolKit* [Sha93a] is a programming toolkit using a client-server based shared memory abstraction.

The *Network Graphics Framework (NGF)* [Sch99a] is an adaptive framework for transmitting 3D graphics over networks. Like our approach, it considers several properties, like capabilities of network, server and client or user preferences. It is not only designed for virtual environments but for general transmission of 3D graphics over networks.

Benford et al [Ben01a] present some general research on virtual environments; Meehan [Mee99a] presented a survey of existing virtual environments.

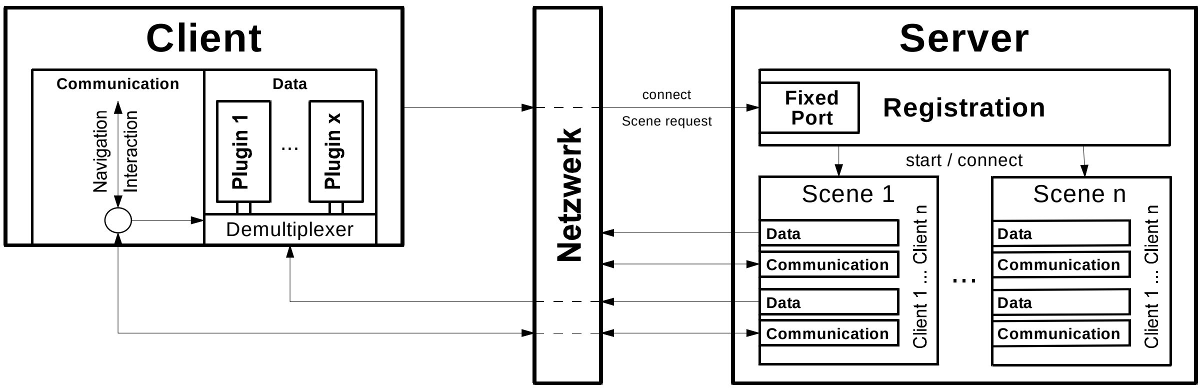

Figure 1: Overview of the framework's architecture

2. REQUIREMENTS

In this section we identify the requirements for the client-server software framework. On an abstract level we describe what the software framework should be able to do and why it should do it.

Generally, we have the following situation: There is an existing 3D world, represented in an arbitrary data format. Furthermore, there is software, that can handle the 3D world data, e.g. read it from a database, and there is software that can handle the visualization and the interaction of the users.

To create a complete 3D client-server environment, the software framework should fulfill the following requirements: It should transfer scene data to the client and it should be able to transfer messages and control information between client and server and between different clients. The framework should also be able to handle multiple 3D scenes and multiple users.
To achieve this, the following properties of the client-server software framework have to be discussed:

Scalability: The server software should be scalable in terms of hardware resources and in number of supported scenes and clients. It should support multi-processor hardware and multiple network connections. The number of supported scenes and clients should be only limited by hardware.

Supported Objects: The client-server framework should be independent of a certain data format or a certain data structure for representation of the scene. The interface to the client-server framework should be flexible enough to support various kinds of information.

Interactivity: The client-server framework should support interactive 3D scenes. This comprises navigation and interaction between the user and objects in the scene, like collision detection, creating new objects or removing objects. Furthermore it should be possible to implement application specific interaction.

Client Adaptivity: The client-server framework should support techniques to adapt to the available resources, like network bandwidth and computing power, optimally.

3. SOLUTION

In this section we describe the solutions of the requirements that we defined in the previous section. The solutions will be general solutions, which means that they will not depend on a special environment, like programming language or an underlying operating system.

3.1 Overview

Figure 1 shows an overview of our framework. Each box corresponds with one process. The server process receives requests from the network and manages the scene processes. One scene process manages one 3D environment and the users, which are connected to the scene. Each user has its own process, managing the network connection and data transfer using the plugins and the multiplexer. For each object type that can be transmitted to the client there is a plugin. For example, if the 3D world consists of textured triangle meshes, there would be a plugin for transmitting the triangle meshes and a plugin for transmitting the textures. Each plugin creates a data stream that is mixed by the multiplexer to one data stream. The multiplexer can be configured that the transmission of an triangle mesh and the according texture end at about the same time. If the amount of data of the texture is bigger than the amount of data of the triangle mesh, then the multiplexer will transmit more texture data in the outgoing data stream. To have easier access to the data of the 3D world, the plugins and the multiplexer should be represented by threads.

On the client side, there is one process managing the network connection, the demultiplexer and the client plugins. The incoming data stream will be demultiplexed and distributed to according plugins in order to get the original data objects. After that the application, using the client-server framework, is responsible to visualize or to process the data objects.

3.2 Connecting to the server

The server process receives a request from the network. Like a web browser always connects the web server at port 80, the connection will be also addressed to a fixed port. The request contains an identifier for a certain scene. The server process checks this request and returns a new connection, connecting the client to the process, which handles the requested scene. If the process for the scene is not running, the server process will start it at first. When the client connects to the scene, it will transmit information about its capabilities over the communication connection. In addition the server starts a bandwidth test in order to determine the capabilities of the network between server and client. With this information the multiplexer on server side and the demultiplexer on client side can be configured optimally. After this configuration the transmission is started.

These are the tasks of the server process:

- Receive requests from the network
- Check the requests
- Dispatch the request to the according scene process
- Start the scene process with the first request
- Manage all scene processes

As concurrency strategy we choose a thread-per-request strategy. Hu and Schmidt [Hu99a] discuss different strategies for web server. A request at the server creates a thread that processes it. This thread checks, if a valid scene is requested and dispatches the request to the appropriate scene. If the scene is not running, which means the user is the first user that requests that scene; this thread will start a new process running the requested scene. Since the server process manages all running scenes it gets a reference to the newly started scene. After the scene is running and the user is successfully connected to the scene, the thread terminates. Then the scene process starts a new thread for each user. This thread first receives the information about the client's capabilities and then performs a network bandwidth test. These informations are needed to load the plugins dynamically, because only those plugins will be loaded, which transmit data according to the client's capabilities. For example, if the client cannot display textures, the plugin for transmitting textures is not be loaded, because textures are not transmitted. The information about the client's capabilities and network bandwidth are also needed to configure the multiplexer optimally. Information for demultiplexing is included in the data stream, transmitted to the client. So the demultiplexer does not need to be configured.

In our implementation we also use this information to optimize the data stream in the following way: The geometric data (triangle meshes) and the textures are stored in a progressive data format. This means, that a basic representation, which is rather coarse, will be transmitted first followed by refinements of the basic representation. Knowing the capabilities of the client means:

- The transmission can be stopped if the display resolution of the client is reached.
- The visual representation can be optimized. If the client is not able to display details of an object as a triangle mesh, the details can be substituted with a texture.

This optimization requires among other things a special representation of the 3D World and some logic to configure the multiplexer. To keep the client-server framework flexible and usable for different kind of 3D world representations, these optimizations are not within the scope of the framework.

3.3 Communication

The task of the communication connection is to transmit information between:

- The client and the scene, e.g. navigation
- The scene and the client, e.g. collision detection
- Two or more clients, e.g. chatting
- Client and multiplexer, e.g. configuration

To accomplish this task, a message consists of an address of a receiver and the message itself. Like the plugins, the communication module runs as an extra thread in the scene process, respectively in the client process. It dispatches the message according to the address the message contains. It is application-specific, interpreting the content of the message.

3.4 Summary

Here we summarize our solution, to meet the requirements, defined in the previous section.

Scalability: In the area of server, multi-processor hardware is a often used approach to increase the computing power by hardware. Our client-server framework is designed to use multi-process and multi-thread techniques. So it is able to completely utilize multi-processor hardware. On single-processor hardware, this design will result in a better utilization of the resources provided by the operating system.

Supported Objects: To be independent of a special data format, we use the concept of plugins. Plugins have a fixed defined interface and are dynamically loaded at runtime. There is plugin for each data, respectively object type that will be transmitted from server to client. This concept has the advantage that the priority of certain object types can be adjusted. This gives the opportunity to optimize the transmission of data.

Interactivity: Interactive 3D scenes are only possible, if information of user input, like navigation, can be transmitted back to the scene. The client-server framework provides an extra communication connection for this. The content of the transmitted message has to be interpreted by the application to keep the over-all design of the framework as flexible as possible.

Client Adaptivity: The client-server framework offers a network bandwidth test, to provide informations that can be used for adapting the data transmission. Additionally the client can offer information about its capabilities to optimize the adaptation. Nevertheless, the actual adaptation has to be integrated in the complete application using this framework.

4. CONCLUSION

In this paper, we presented a framework for client-server based, interactive, 3D environments. The framework is designed for scalability, it is independent of the data format for the 3D world, it supports communication between all parts of the system and it offers possibilities for optimizing the data transmission. The framework is suitable for powerful workstation as client and is also suitable for smaller devices. The described framework is used in the project WAP for graphical objects funded by the Heinz-Nixdorf-Foundation, basing on the real-time and co-operative visualization of large, dynamic, and interactive 3D scenes on different, possibly mobile devices.

There are still issues that can be improved. Scalability could be improved by distributing the server part of the application on a cluster of computers. Since the scene processes do not share any data with each other and are only loosely connected to the server process, it might be possible to run them on different computers. Issues regarding security are also not considered yet. When transmitting information using the internet, security functions might be useful to avoid misuse.

5. REFERENCES

[Ben01a] S. Benford, C. Greenhalgh, T. Rodden, J. Pycock, "Collaborative Virtual Environments", Communications of the ACM}, Vol. 44, No. 7, 2001.

[Car93a] C. Carlsson, O. Hagsand, "DIVE - a Multi-User Virtual Reality System", IEEE Virtual Reality Annual Symposium, 1993.

[Fun96a] T. Funkhouser, "Network Topologies for Scalable Multi-User Virtual Environments", IEEE Virtual Reality Annual International Symposium (VRAIS '96), 1996.

[Hu99a] J. Hu, D. Schmidt, "JAWS: A Framework for High-performance Web Servers", Domain-Specific Application Frameworks: Frameworks Experience by Industry, M. Fayad and R. Johnson (Editors), John-Wiley, 1999.

[Lea97a] R. Lea, Y. Honda, K. Matsuda, O. Hagsand, M. Stenius, "Issues in the Design of a Scalable Shared Virtual Environment for the Internet", Proceedings of the HICSS '97, 1997.

[Mac97a] M. Macedonia, M. Zyda, "A Taxonomy for Networked Virtual Environments", IEEE Multimedia, Vol. 4, No. 1, 1997.

[Mac94b] M. Macedonia, M. Zyda, D. Pratt, P. Barham, S. Zesswitz, "NPSNET: A Network Software Architecture for Large Scale Virtual Environments", Presence, Vol. 3, No. 4, 1994.

[Mee99a] M. Meehan, "Survey of Multi-User Distributed Virtual Environments", Course Notes: Developing Shared Virtual Environments, SIGGRAPH 99, 1999.

[Sch99a] B. Schneider, I. Martin, "An Adaptive Framework for 3D Graphics over Networks", Computers \& Graphics, Vol. 23, No. 6, 1999.

[Sha93a] C. Shaw, M. Green, J. Liang, Y. Sun, "Decoupled Simulation in Virtual Reality with the MR Toolkit", ACM Transactions on Information Systems, Vol. 11, No. 3, 1993.

[Sno94a] D. Snowdon, A. West, "The AVIARY VR-System. A Prototype Implementation", 6th ERCIM Workshop, 1994.

[Wat96a] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, W. Yerazunis, "DiamondPark and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction and Runtime Modificability", Mitsubishi