A Framework for Video-based and Hardware-Accelerated Remote 3D-Visualization

Frank Goetz, Gitta Domik

Computer Graphics, Visualization and Image Processing
Fuerstenallee 11, D-33102 Paderborn, Germany
(frank.goetz|domik)@uni-paderborn.de

Abstract

This paper presents a framework for video-based and hardware-accelerated remote 3d-visualization that produces and delivers high quality video streams at an accurate frame rate. The framework is based on the portable scenegraph system OpenSG, the MPEG4IP package and the Apple Darwin Streaming Server. In realtime generated visualizations will be multicast as an ISO MPEG-4 compliant video stream over the RealTime Streaming Protocol from a server to a client computer. On the client computer a Java program and an ISO MPEG-4 compliant video player are used to interact with the delivered visualization. While using MPEG-4 video streams we can achieve high image quality at a low bandwidth.

Categories and Subject Descriptors (according to ACM CCS):
I.3.1 [Computer Graphics] Distributed/network, C.2.4 [Distributed Systems]: Distributed applications.

1. Introduction

Not so long ago the lack of communication was hampering the progress of research. Today, researchers share information over the internet and cooperation is demanded in all areas of the industry. Particularly computer-generated visualizations took a firm place in many fields, like geosciences, medicine, architecture, automobile construction and space technology.

Interactivity and dynamics in realtime generated visualizations were strongly improved by today's more efficient graphics accelerator boards and arising multiprocessor systems. Often complex and interactive visualizations can only be rendered on powerful graphics servers or graphics clusters, since only they achieve the desired representation speed and image quality. Therefore it is necessary to transfer the power of these graphics servers to the location and the computer of the user that wants to work with this visualization. In the last years scientists and researchers have given a great deal of attention to the area of remote visualization.

A common method to transport visualizations is based on the streaming of compressed meshes and textures from a server to a client computer. While visualizing complex scenes with a huge amount of polygons this method often needs too much network capacity and has high demands on the capabilities of the client computers. In most cases, the client computer offers only a fragment of processor power and graphics power that a server is able to achieve. Finally, with this method it is not possible to get accurate frame rates while sending interactive and realtime generated visualizations from a server to a client computer. Accurate frame rate means at least a frame rate of 20 frames per second. In that case a user has to accept limitations because compromises were made by sending large datasets to the client computer.

The aim of our work is to develop a platform independent visualization server that takes advantage of the functionality of current graphics accelerator boards and delivers a high performance video stream to the remote or client computer. In our case high performance video stream means that the picture quality is high, the frame rate is accurate, the needed bandwidth is low and the

latency is as short as possible. While using video streams we have a constant and calculable size of network capacity and processor power that has to be supported by the client computer. The size of the video stream is independent from the size of the visualization and the graphics accelerator board of the client computer does not need special features to display interactive and animated visualizations at an accurate frame rate. It is possible to use the majority of available computers, even laptops and handhelds, as a visualization client.

A further point should be that different users are able to access the data that was generated by the visualization server at the same time. Also the coordination between active users should be as easy as possible. It is very important that every user of the cooperative working process gets the same potential for work. There should not be any problems concerning the geographical location or the capabilities of the hardware. The user should not be restricted or excluded from the working process because of using limited hardware or software. The ability to interact, navigate and manipulate at an accurate speed has to be available for all users of a cooperative working process.

The rest of the paper is organized as follows: first, other frameworks and solutions for remote 3D-visualization are introduced; then all necessary packages, libraries and servers that we need for our solution are presented; afterwards, the architecture of our system is explained; in the last part results are presented.

2. Previous Work

There are different ways of realizing remote visualization. One of the latest remote visualization frameworks is Chromium[1]. Chromium is a stream-processing framework for interactive rendering on clusters.

Silicon Graphics, Inc. provides a commercial solution called OpenGL Vizserver[2]. OpenGL Vizserver is a technical and creative computing solution designed to deliver visualization and collaboration functionality to any client, whether on a desktop workstation or a wireless tablet. OpenGL Vizserver allows users to remotely view and interact with large data sets from any other system at any location in an organization and to collaborate with multiple colleagues using these same applications and data. Because of design decisions the OpenGL Vizserver works only with the SGI Onyx family. Similarly to the solution of Stegmaier et al.[3], the VizServer relies on dynamically linked executables in order to be able to implant its functionality without modifying the target application.

The *generic solution for hardware-accelerated remote visualization* from Stegmaier et al.[3] works transparently for all OpenGL-based applications and OpenGL-based scene graphs and does not require any modifications of existing applications. They use a similar approach as Richardson et al.[4] in their paper *virtual network computing*. In this paper a remote display system that does not support the remote use of 3d graphics acceleration hardware is presented.

Ma and Camp[5] developed a solution for remote visualization of time-varying data over wide area networks. This system involves a display daemon and a display interface. The data from the renderer is automatically compressed, transported and decompressed. By using a custom transport method, they are able to employ arbitrary compression techniques.

Other solutions that support remote visualization were developed by Engel and Ertl[6] and Engel et al.[7,8]. In *texture-based volume visualization for multiple users on the world wide web* Engel and Ertl describe a volume visualization tool that uses JAVA and the Virtual Reality Modelling Language (VRML). In this case the client computer has to be equipped with a 3d graphics acceleration hardware to render the transmitted VRML scene with an accurate speed. In their latest papers Engel et al. describe visualization systems that use image compression technologies to transport the visualization data from the server to the client computer. In their paper *a framework for interactive hardware-accelerated remote 3d-visualization* the visualization parameters and GUI events from the clients are applied to the server application by sending CORBA (Common Object Request Broker Architecture) requests.

Their remote visualization framework is very similar to our framework. Instead of CORBA we developed our own network communication layer. The main difference to all other remote visualization frameworks and also the demarcation to the systems by Engel et al. is that our framework transports MPEG-4 video streams instead of sending only pictures from the server to the client. This improves the quality and capacity of the data to be sent.

3. Background for our solution

One requirement of our framework is to use only free and non commercial packages, libraries and servers. In this chapter all used packages, libraries and servers are presented briefly.

3.1. CommonCPP

CommonCPP offers a highly portable C++ application development framework. It provides classes for threads, sockets, daemon management, system logging, object synchronization, realtime network development, persistent object management, and file access. These are all features that are not supported by standard C++ libraries.

The CommonCPP Framework is used by our CommServer for the communication between the server and the client. The CommServer is a socket based and thread per session server. For every session a new thread will be generated. All incoming messages will be processed by this thread and after the connection is closed the thread will be terminated. The CommServer listens on a standard port and generates a NetRPCMgr object for incoming connections. Another port is used for the data transportation between the CommServer and the client. After a connecting admission the NetRPCMgr object waits for further connections. The NetRPCMgr reads all incoming data and sends them as a byte-array to the ClientHandler. The ClientHandler tries to analyze this data. If the type of the data structure was detected, all possible parameters will be extracted. With these parameters the fitting server functions can be called. In this way all interactions that are initiated by the client computer can be managed.

3.2. OpenSG

OpenSG[9] is a portable scenegraph system to create realtime graphics programs. OpenSG runs on different platforms like IRIX, Windows and Linux and is based on OpenGL. OpenSG is not a complete VR system. It is the rendering basis on top of which VR systems can be built. OpenSG has a nice list of supported features, some of which are unique, and thus make OpenSG a very useful scenegraph.

Multithreaded asynchronous scenegraph manipulation is one of the central parts of the OpenSG design. The OpenSG data structures are set up in a way that allows multiple independent threads to manipulate the scenegraph independently without interfering with each other. This feature is very interesting and important for remote visualization systems that should support collaborative work. By using this feature the manipulations of each user will be synchronized with the manipulations of the other users. Finally, every user of the collaborative working community gets the same view on the current dataset.

3.3. Video4Linux Loopback Driver

This driver implements a video pipe using two video4linux devices. The first device is used by the program supplying the data. The second device acts as a normal video4linux device, it should be usable by any application that fulfills the video4linux specifications.

The loopback device has two operating modes:

In the simple *one-copy mode* the supplying program specifies the size of the images and the used palette and uses the write function to push its images to the pipe. This mode is mostly for feeding fixed size images without any knowledge about the client. At the moment our system only supports this mode and streams a 24bit true colour video with a size of 352*288 pixels to every client computer.

In the *zero-copy mode* the supplying program regularly polls the device. Here the supplying program has almost complete control over the device's behaviour and it will be mainly used to implement complex multiple tuner, channel and size configurations. This mode is of interest for allowing the remote client to change the resolution of the streamed video. Especially for computers with small displays like handhelds it is important to reduce the size of the video stream.

3.4. MPEG4IP

The MPEG4IP project[10] was started by Cisco's Technology Center to further the adoption of audio and video streaming standards, and to serve as a toolkit to track the ISMA (Internet Streaming Media Alliance) specifications and requirements. MPEG4IP provides an end-to-end system to explore MPEG-4 multimedia. The MPEG4IP package includes many existing open source packages and offers the possibilities of integrating them together. This is a tool for streaming video and audio that are standards-oriented and free from proprietary protocols and extensions. MPEG4IP is available for Linux, Windows and MacOS. In our Framework we use two components of the MPEG4IP package:

The *MP4Live Server* is able to produce MPEG-4 conform video streams in real time. The basic idea is that the video stream will be supplied by a video capture board or a camera. Afterwards it will be compressed and transmitted to another computer. At the moment the MP4Live Server is only available for Linux, the rest of the MPEG4IP package is available for Windows and MacOS, too. Until now only the open source and ISO MPEG-4 compliant XviD (http://www.xvid.org) video codec is supported by the MP4Live Server, but without problems other video codecs can be adapted for the MP4Live Server.

The *MP4Player* supports different video/audio codecs and the RTP/RTSP (Realtime Transport Protocol / Real-

RealTime Streaming Protocol) protocols. We use the source code of the MP4Player to build our own dynamic link library. This library represents the non Java part of our client software.

3.5. Apple Darwin Streaming Server

The Apple Darwin Streaming Server is a server technology (http://developer.apple.com/darwin) which allows sending streaming video data to clients across the Internet using the industry standard RTP and RTSP protocols. Further on, the Apple Darwin Streaming Server supports a new technology called Instant-On Streaming. Instant-On dramatically reduces the delay caused by buffering of the media stream prior to playback. Another important feature is the MPEG-4 support. The Apple Darwin Streaming Server can serve ISO-compliant MPEG-4 files to any ISO-compliant MPEG-4 client, including any MPEG-4 enabled device that supports playback of MPEG-4 streams over IP. A computer can serve on-demand or live MPEG-4 streams. Finally, the Apple Darwin Streaming Server supports a new and interesting feature called Skip Protection. Skip Protection uses excess bandwidth to buffer available data faster than real time on the client machine. The communication between client and server results in retransmission of only the lost packets, in case packets are lost. This reduces the traffic in the network.

4. Implementation

4.1. Server

Our framework consists of three different servers: the Apple Darwin Streaming Server, the MP4Live Server and our Visualization Server based on OpenSG (see Figure 1). The principal reason for using OpenSG was the support of clustering. So in the future our visualization framework can easily be extended to be used by several computers at the same time. Then as many different views on a dataset can be delivered as computers in the cluster are available and every user can change his line of sight explicitly. Momentarily only one user has the possibility of changing the line of sight at the same time. All other users have to accept this choice. When using a cluster, all three before mentioned servers have to be installed on each computer of this cluster. Each cluster node needs a powerful graphics accelerator board, enough memory and processor power. The different Darwin Streaming Servers and MP4Live Servers supply for each user a view-dependent video stream.

At the moment we can only use Linux as a server system, because the MP4Live Server is not available for Windows or MacOS. Furthermore, a video device has to be installed for the MP4Live Server. This can be done by using the Video4Linux loopback driver. Our visualization server provides this new virtual video zation server provides this new virtual video device with the current rendered OpenGL Frame.

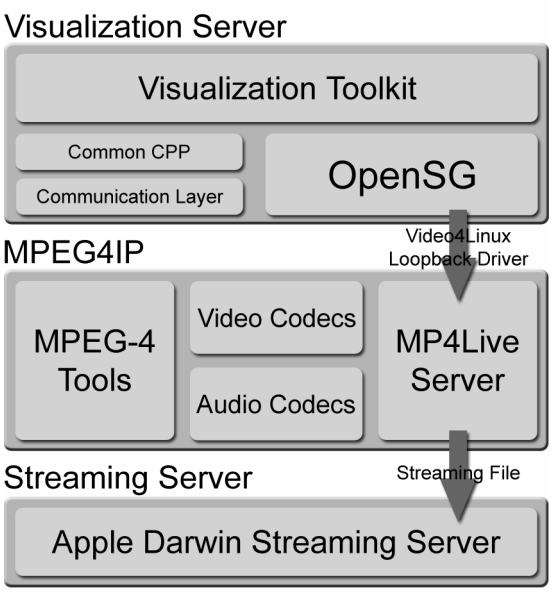

Figure 1: Server architecture

In the last years, the grabbing of the OpenGL framebuffer was a large problem when using consumer graphics accelerator boards from companies like NVIDIA or ATI. Only high-end graphics computers for example from SGI supported fast reading out of the framebuffer of the graphics accelerator board into the main memory of the computer. To ensure that the OpenGL framebuffer is not empty, because the OpenGL window is covered by another window, we grab the backbuffer instead of the frontbuffer. This does not work perfectly with all drivers. On our system even the screensaver can turn on without causing problems to the OpenGL rendering process. Another possibility is the use of the pbuffer (preserved pixel buffer), a hardware-accelerated off-screen buffer. Unfortunately not all drivers on different platforms support preserved pixel buffers. The easiest way would be to use the accumulation buffer, but most graphics boards or drivers have no hardware-accelerated support for this buffer. Without hardware acceleration the use of the accumulation buffer is ten times slower than the use of hardware-accelerated buffers.

At the moment our visualization server grabs 50 frames per second and sends them to the video device. The MP4Live Server uses the video device as input device and generates a MPEG-4 video stream. The frequency of the encoded video is 25 frames per seconds. The

MP4Live Server supports unicast or multicast video streams. Until now the encoded MPEG-4 video stream is only available for the server. Finally, we need a streaming server that distributes the video stream in the network. We decided to use the Apple Darwin Streaming Server, because it fulfills all our requirements. After the installation and configuration of the Apple Darwin Streaming Server every client computer with installed ISO-compliant MPEG-4 video player is able to display the generated visualization.

4.2. Client

Our goal is to realize the client software completely in Java. Such a solution is platform independent and runs on arbitrary operating systems and hardware platforms, like for example handhelds, notebooks and Tablet PCs. Currently most parts of the client software are written in Java (see Figure 3), like the whole communication layer and the Graphical User Interface. The communication layer of the client works exactly like the in C++ written CommServer, but was completely designed in Java. Nowadays a complete Java Runtime Environment is installed on the majority of client computers. For this reason we decided to build the graphical user interface with Swing instead of using the rudimentary AWT Toolkit. The graphical user interface is very flexible, the appearance of all buttons, menus and their functionality are specified on the server. If the server delivers new functionality, the status of a user changes or a new visualization with different features is loaded, then the client software has not to be reloaded. Only the entries in the menus and the appearance of the buttons will be changed.

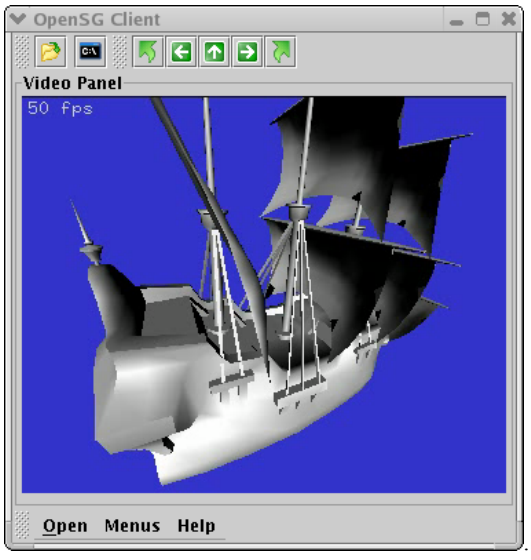

Figure 2: Player library embedded in the client software

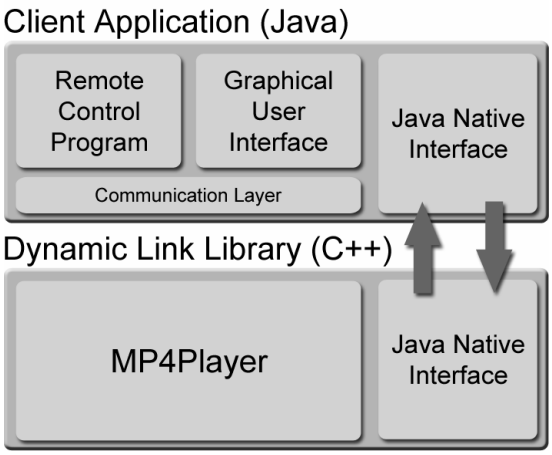

Figure 3: Client architecture

At the moment there are three different kinds of user categories:

a) a user that has full control and is able to navigate into the generated scene
b) a user that has full control, but can only manipulate and not navigate in the rendered scene
c) a user that is only allowed to watch the visualization, but has not the possibility of changing anything

If the MP4Live Server uses the multicasting protocol many users can be of the category two and three, but only as many users as computers are available in the cluster can be of the category one. While using the multicast protocol instead of a normal unicast protocol the server does not need more calculation power to provide all users with a video stream. Only the routers have to distribute more data packages.

To deliver the whole functionality of the server to the client computer every visualization technique of the framework and all functions of the visualization framework itself have to offer an interface to receive and send all important parameters from the server to the client and vice versa. The values of these parameters can be modified by the user at the client computer. For the mouse navigation and interaction in the rendered scene all parameters have to be sent from the client computer to the server by using the existing socket connection. Furthermore all menus, toolbars and popup windows offer functionality to control the specified visualization technique and settings to define the appearance of the different visualizations. The server program has the ability to change the look and feel of the graphical user interface at the client side. Thereby new functionality in the server program can be made available to the user at

run-time without an explicit update of the client software.

To view the video stream the user has two possibilities. The first one is to download a dynamic link library based on the MP4Player. This library is written in C++. All communications between the client software and the video player library are done by the use of the Java Native Interface. The player library generates a frame with the video stream that is delivered from the server. This frame can be integrated in our Java based client software as shown in Figure 2. If the user does not want to download anything there is the possibility of viewing the delivered video stream with a common MPEG-4 video player (see Figure 4). This is possible because the Apple Darwin Streaming Server delivers an ISO-compliant MPEG-4 stream over the RTP/RTSP protocol. Tests with the mp4player provided with the MPEG4IP package, the Apple Quicktime 6 Player and the RealPlayer with Envivio plug-in delivered positive results. Thus, in principle, all RTSP capable video players, which support an ISO-compliant MPEG-4, can be used.

In a future version we will support the Java Media Framework. At the moment there are no ISO-compliant MPEG-4 codecs that can be used with the Java Media Framework and that support the RTSP protocol. The *MPEG-4 Video for JMF* plug-in from the IBM alphaWorks group is one of the first pure Java MPEG-4 solutions for the Java Media Framework, but by now this plug-in is only able to display video streams that were created with the MPEG-4 Simple Profile. At the moment we are working on embedding the XviD video codec into the Java Media Framework.

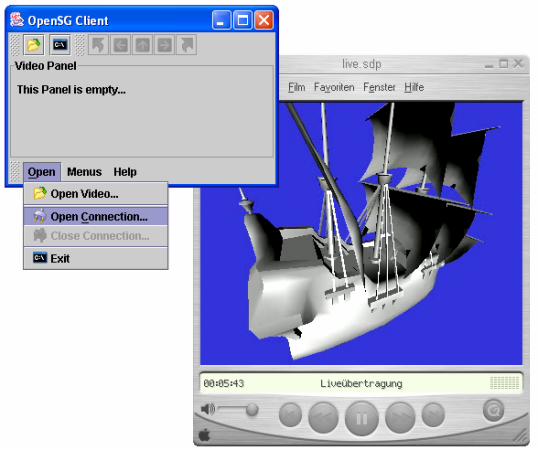

Figure 4: Client software with Quicktime 6 Player

5. Results

In the following tests we took a Dell Server with 2.4 GHz Pentium4 processor, 1024 MB main memory and NVIDIA GeForce4 TI graphics accelerator board. A Dell Laptop with a 1.4 GHz Pentium4 mobile processor, 512 MB main memory and a NVIDIA GeForce4 440 Go graphics accelerator board acted as client.

At a resolution of 352*288 pixels our server needs only less than 40 percent of the processor time to calculate the MPEG-4 video stream with the XviD codec. The rest of the time can be used to render the visualization. If we change the resolution to 768*576 pixels, the calculation time for the video stream needs nearly 70 percent of our processor time. More results can be found in Table 1. Other resolutions are at the moment not interesting, because a limited client computer would not have the processor power to decode such video streams.

Resolution	Format	Used processor
128*96	SQCIF	7% – 8%
176*144	QCIF	11% - 12%
320*240	SIF	31% – 32%
352*288	CIF	36% - 37%
352*480	Half D1	48% – 49%
640*480	4SIF	63% - 64%
704*480	D1	64% - 65%
720*480	NTSC CCIR601	65% - 66%
768*576	SQ PAL	69% - 70%

Table 1: Processor time for encoding the video stream

Changing the kilobit rate from 100 Kbit/s up to 4000 Kbit/s has no effect on the calculation time. Only the image quality and the capacity of the needed bandwidth increase. The different qualities of the different compression factors are shown in Figure 5.

Next we will summarize some results concerning the glReadPixels function. With an OpenGL window resolution of 640*480 we lose on a Geforce4 TI 4200 graphics accelerator board at the most a quarter of the frame rate while using the glReadPixels function. In our special case the frame rate changes from 80 to 60 frames per second. At the same resolution, the frame rate decreases to half as much on a GeForce2 MX graphics accelerator board while using the glReadPixels function. An interesting phenomena is that on a GeForce4 440 Go graphics accelerator board we hardly lose any frames per second while using the glReadPixels function.

Finally, we will give some information about the latency. In a local area network with a 100 Mbit/s Ethernet connection we a have a streaming latency lower 0.5 seconds between the server and the client. The communication latency between the server and client is lower then 0.1 seconds.

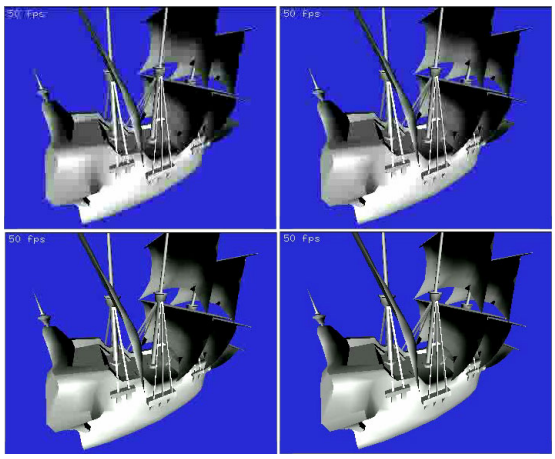

Figure 5: Encoded video frames at a compression rate of 100KBit/s, 1000KBit/s, 2000Kbit/s and 4000KBit/s

6. Conclusions and Future Work

We have presented a framework which allows interactive hardware-accelerated remote 3d-visualization from nearly all Internet-connected desktop PCs. Only a Java Virtual Machine and an ISO-compliant MPEG-4 video player, that supports the RTSP protocol, have to be installed on the client computer. In the near future we will additionally support the Java Media Framework. Then only the Java Virtual Machine and the Java Media Framework have to be installed on the client computer. For the reason that we use MPEG-4 video streams and Instant-On Streaming we get a high picture quality and a minimum of latency. Even connected with an ADSL (Asymmetric Digital Subscriber Line) connection a user can work at an accurate speed on the visualization server.

In future versions of our framework we will improve the collaborative features and the visualization toolkit. Therefore we have to expand our server to more than one computer, we have to connect the individual server computers and we have to ensure that the manipulations of different users will be correctly adjusted to the visualization.

Recapitulatory it can be said that the use of a visualization server offers flexible possibilities for spatially separated cooperation partners that have only a standard PC with a normal internet connection as input device.

Acknowledgements

The authors wish to thank the students of the project group “creativity & technique” for their help on implementing parts of the system.

References

1. G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, J. Klosowski. Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters, SIGGRAPH 2002.
2. Silicon Graphics, Inc. OpenGL Vizserver 3.1: Application-Transparent Remote Interactive Visualization and Collaboration, *Technical White Paper*, Silicon Graphics, Inc., June 2003. http://www.sgi.com/software/vizserver/.
3. S. Stegmaier, M. Magallón, and T. Ertl. A Generic Solution for Hardware-Accelerated Remote Visualization. In *Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym '02*, 2002.
4. T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1), January 1998.
5. K.-L. Ma and D. M. Camp. High performance visualization of time-varying volume data over a widearea network status. In *Supercomputing*, 2000.
6. K. Engel and T. Ertl. Texture-based Volume Visualization for Multiple Users on the World Wide Web. InGervautz, M. and Hildebrand, A. and Schmalstieg, D., editor, *Virtual Environments ’99*, pages 115–124. Eurographics, Springer, 1999.
7. K. Engel, O. Sommer, and T. Ertl. A Framework for Interactive Hardware Accelerated Remote 3DVisualization. In *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym ’00*, pages 167–177,291, May 2000.
8. K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt, and T. Ertl. Combining Local and Remote Visualization Techniques for Interactive Volume Rendering in Medical Applications. In *Procceedings of IEEE Visualization ’00*, pages 449–452. IEEE, 2000.
9. D. Reiners , G. Voß G., Behr J., 2002: OpenSG - Basic Concepts, In: *1. OpenSG Symposium, Darmstadt*, Germany, 2002
10. D. Mackie. Streaming Video & MPEG4IP. *Presentation of MPEG4IP at the Silicon Valley Linux User's Group on February 6th*, Cisco Technology Center, 2002.