Fourth Eurographics Workshop on Parallel Graphics and Visualization (2002)
D. Bartz, X. Pueyo, E. Reinhard (Editors)

An Efficient System for Collaboration in Tele-Immersive Environments

N. Jensen[1,2], S. Olbrich[2], H. Pralle[2] and S. Raasch[3]

[1]Learning Lab Lower Saxony (L3S), Germany

[2]Regionales Rechenzentrum für Niedersachsen (RRZN) /
Lehrgebiet Rechnernetze und Verteilte Systeme (RVS), University of Hannover, Germany

[3]Institut für Meteorologie und Klimatologie, University of Hannover, Germany.

Abstract

The paper describes the development of a high-performance system for visualizing complex scientific models in real-time. The architecure of the system is a client/server model, in which the simulator generates lists of 3D graphics objects in parallel to the simulation, from where they are sent to a streaming server. The server transfers the 3D objects to viewer clients. Clients communicate over a second connection with each other, which adds the ability to perform collaborative tasks. An application related to computational fluid dynamics is specified where such a tele-immersive system can be used. The approach differs to other solutions because it offers a large set of graphics primitives for visualization, and it is optimized for distributed, heterogenous environments.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems

1. Introduction

The paper describes the development of a network-distributed visualization system for interactive viewing and steering of scientific models. The system visualizes a simulation parallel to progression, supports interactive, collaborative exploration (CSCW), and computational steering. Three major differences to other systems are: the software and the format of data are optimized for speed, and cover a wide range of 3D visualization primitives. Second, the system makes it possible to explore 3D scenes simultaneously to simulation, without access to discs or tapes, even when most other solutions fail due to data overload. Third, in contrast to scalable systems that are optimized for shared memory architectures, the system presented in this paper is used in environments with a combination of shared and distributed memory.

Design and implementation of such a visualization system is a challenge because data that result from computational simulations on supercomputers approach the Terabyte limit. Such volume cannot be accessed in real-time, and worse, the amount of visualization data necessary to produce images adds to the complexity. Data distribution and reduction are important factors.

We can categorize data in two groups. Simulation data are generated as a result of an algorithm that implements a model of a physical process. They represent the largest part of data that a computational simulation produces. We cannot reduce the data set, however, because it is not controlled by the visualization system. For instance, data can be erased immediately after analysis, or stored. If they are stored they may or may not be postprocessed. The number of combinations makes it impossible to imply characteristics of simulation data. Therefore, when we discuss techniques for distributing and reducing data with respect to a versatile visualization framework we limit our scope to visualization data and the optimization of processes that operate on them. Visualization data belong to the second category of data. Operations on them can be applied, similar to those above.

Visualization data are part of the visualization process that generates a sequence of visible images from simulation data in four stages. These four constitute the visualization pipeline [20]:

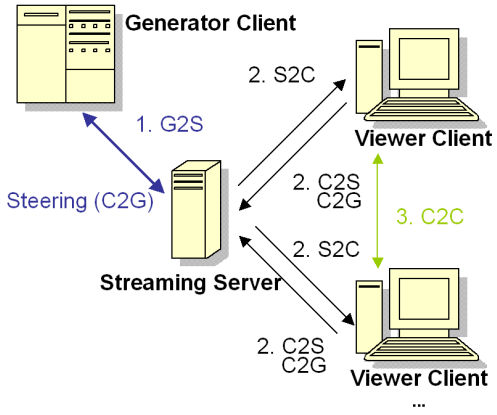

Figure 1: *Distributed topology of the DocShow-VR streaming system.*

1. Filter, i.e. the selection and modification of generated simulation data.
2. Map, i.e. the generation of descriptions of graphics data from filtered data. Graphics data describe geometries and visual properties.
3. Render; the transformation of graphics data to an image.
4. Display; presenting the image to the user.

The system described in the paper is a software that implements the visualization pipeline, see Figure 1 (an explanation follows in section 3). For each stage of the pipeline there is one component. The components can be distributed among heterogenous computers. Some benefits are that components are loosely coupled, and second, users can make optimal use of their pool of hardware to equal workload balancing to minimize idle time, and they can exploit the strengths of heterogenous hardware.

The paper delineates previous work in section 2. Section 3 is an overview of the system and discusses ideas that have influenced the design. We present preliminary extensions for tele-immersive collaborative virtual environments in section 4, and draw some early results. Finally, we summarize, and address future work.

2. Previous Work and Shortcomings

Many commercial packages visualize numerical scientific simulations, see [6]. A discussion of each system is out of the scope of the paper, but we can summarize their shortcomings in one paragraph.

- **Monolithic hardware:** only one computer can be used for visualization: all steps of the visualization pipeline are carried out by one machine. This is not only inefficient because each stage relies on different hardware features that a single hardware architecture does not provide optimally all in one, it implies that each user has access to such a system.
- **No distribution:** visualization requires a large set of software modules installed on a computer, often controlled by a GUI, and an integrated programming environment. The components cannot be distributed in a network of specialized hardware to exploit the individual strengths of each component. It is difficult to balance workload between hardware, in order to minimize idle time.
- **No efficient option for filtering and mapping:** the user cannot access a library for filtering and mapping, or it does not support parallel programming to generate visualizations efficiently.
- **Huge data load:** the system follows an all-or-nothing strategy: it can only visualize a complete set of simulation data. This increases effort to store data if the results outreach several hundreds of Gigabyte. At least, interactivity and response time of the system are negatively affected. Worst, data size exceeds capacity of hardware (discs, networks).
- **Limited performance:** the algorithms are not optimized for speed. Data structures contain considerable overhead, e.g. ASCII representation, or tree structures of data, instead of lists. This restricts complexity of 3D scenes that can be processed in an efficient way.
- **Batch-mode visualization:** visualization and simulation are separate processes. The user has no interactive, intuitive method at hand to steer the simulation in response to visual output.
- **Tight coupling:** if a set of images of 3D models is stored that has been generated by a simulation, and a user wants to change rendering parameters, she would need to restart the simulation to update the sequence of images.
- **Not enough visualization methods:** some systems implement very sparse sets of visualization methods, e.g. only 2D slicers. This restricts effective visualization.
- **No integrated option for collaborative working:** Collaboration between peers is limited, such that the number of collaborators is restricted to those that can physically meet around a computer or can use supplementing media.

Some research prototypes address these problems. An overview of representative immersive virtual environments specify [8]. However, these systems do not process time-variant very large data sets (Terabyte). Steering and collaboration in high performance applications on middleware are discussed in [13]. In contrast, our system does not make use of intermediary protocols or data structures, to minimize overhead. [7, 1] have developed a system that aims to tackle the problem of large data visualization on top of their system, VTK. Their "demand-driven" approach reduces transfer of data effectively in some cases, but at the cost of implementing a complicated update policy that requires viewer clients to trigger update commands. The solution offers 3D streaming, as we do, but splits up objects and streaming channels among processors in a distributed pool of computers

to balance workload. Their policy of decomposition leads to problems at boundary regions between domains, which must be resolved explicitly. Because our aim is to support time-variant simulations with changes in topology, our system updates graphics automatically, and avoids streaming decomposition, which is beneficial if topology of visualization data changes frequently. Parallel filtering and mapping with our mapping library is application-driven, and transparent to viewer clients.

The system described in the paper shares characteristics with SGI VizServer [16]: remote rendering with OpenGL, use of heterogenous hardware, and support for collaborative visualization. The difference is that VizServer separates rendering (server) from display (client). DocShow-VR separates generation of graphics data from rendering. Rendering and display are performed on clients. The policy is more advantageous if clients must render different views independently from each other. VizServer, on the other hand, is more suitable if such requirements do not exist. VizServer is a better solution if client-hardware (including the network) cannot process and render scenes efficiently, or if legacy rendering software must be made available to multiple users. VizServer is designed for exclusive interoperability with SGI hardware. DocShow-VR does not make this restriction. A combination of both systems would allow users to customize the system in a very flexible way and improve adaptability to hardware settings. However, this is subject of future work.

In contrast to comparable systems that support volume rendering [4, 3, 11, 14] our implementation focuses on interoperability with standards for hardware acceleration and polygon-based rendering. Image-based rendering, which simulates volume rendering by interpolating between images, is not used because it generates visual artifacts, cannot be as easily analyzed by algorithms as polygons, and provides fewer rendering options on side of the client.

An earlier version of our system is described in [9]. Due to its real-time streaming mode, the software can be used in scenarios where accumulation of generated data exceeds disc space. Efficient networks (Gigabit) and parallel computing are a pre-requisite for optimal use. Design, implementation, and extensions for computer supported collaborative working are specified in section 3.

Streaming is used by all our system components. The streaming format needs to specify 3D content to permit the application to decouple rendering from filtering and mapping, an option not possible with images. Prior to developing our own streaming format we considered The Virtual Reality Modeling Language [18], and MPEG-4 (BIFS binary format for 3D scenes) [19]. Parsing and transfer of scene graphs in VRML is not efficient due to ASCII representation and because of the complexity of the nested tree structure where descriptors of geometries and materials are arranged. A similar general approach is propagated by the inventors of MPEG-4 which embeds abstract media objects into different stream layers. In summary, present streaming formats can represent complex structures of heterogenous multimedia objects. They are suitable for transferring data on networks with low and medium bit rates. More streaming formats to present 3D content are available, they share similar characteristics, and use data compression. For static scenes that are progressively built this makes sense, but not for animated sequences in which rapid changes in scene topology occur frequently.

3. 3D Streaming with DocShow-VR

3.1. Strategies to Visualize Scientific Models

An approach to make scientific models visible is to create sequences of 2D images and to store them in tertiary memory, e.g. hard discs or tapes. However, this makes it difficult to explore the visualization during simulation, and does not build on the strengths that the use of 3D sequences would offer: flexible choice of rendering options, real-time steering of a simulation driven by users, and interactive exploration of spatial data. On the other hand, a sequence of 3D scenes is sometimes too large to be held in tertiary memory together with simulation data, and to be retrieved from there in real-time.

In addition to the two options above, the system generates the 3D animation frame by frame in parallel to the simulation. One frame contains a complete 3D scene. The server sends each completed scene from main memory to the streaming server that replicates and forwards the stream to each client for rendering and display. Rendering and display could be split up in the same way, but this is not currently supported. There is another reason why it is advisable to decouple rendering from mapping: workload for rendering can be taken from the computer that carries out the simulation, and delegated to machines that are specialized to perform this work efficiently because their hardware architecture is designed for that task. Although the generator could do the rendering, it could do it less efficiently than a graphics station. On the other hand, a graphics station is not designed to perform numerical simulations. For these reasons, our system decouples the visualization process: filtering and mapping are performed on the generator, and rendering and display are carried out on graphics machines. This is an economic approach because users can employ dedicated hardware for each of these steps, hence improve on total cost of ownership and operations/s. The drawback is that Gigabit networks are required, but this problem diminishes as hardware becomes more widespread. The system does not decouple simulation, filtering, and mapping because the amount of simulation data exceeds the amount of visualization data, compare [15]. Two advantages arise from that: first, copying and transfering visualization data, not simulation data, limits workload on the network. Second, filtering and mapping is done by the machine that performs the simulation and can be carried out by parallelized algorithms that are tuned to optimally co-operate with the algorithm that carries out the sim-

ulation. The specific strengths of the machine, in this case parallel numerical computations, can be used. The disadvantage is an increase of workload on the data generator. However, if several clusters are used, a different cluster may be allocated for each set of processes, and access shared memory.

3.2. Design Goals

Based on shortcomings identified in section 2 we formulate design goals.

- **Distributed hardware:** the user can exploit the special features and strengths of distinct hardware, and rely on networks with high bit rates (at least 100 MBit/s).
- **LibDVRP – parallel filtering and mapping:** the user should access a library that makes filtering and mapping easier. Speedup by parallel programming is paramount for many applications.
- **Incremental transfer of visualization data:** physical models are time-dependent. The system can exploit this property in a way that visualization data are sent for each frame to viewer clients. Transfer of sub-frames increases management efforts by algorithms, and is not beneficial for animations.
- **Maximize performance:** all algorithms and protocols are optimized. ASCII representations need to be substituted by binary formats. Data compression is not used because with a steady increase of network capacity the difference between computer and network performance diminishes.
- **Dialog-mode visualization:** computational, interactive steering requires users to communicate with generator clients in a dialog-based fashion. This is not always possible, for instance, if changes of parameters and replay of intervals are not practical on supercomputer simulations because this would imply the use of some kind of roll-back mechanism. If data are not stored, or not stored regularly, such options cannot be provided. Moreover, supercomputer capacities are a precious resource, and seldom they will be driven in dialog-based mode; instead, batch-mode is used. This is different when workstations generate simulation data, where these facilities are present, and steering a practical option.
- **Loose coupling:** 3D streaming enables users to explore scenes under different viewpoints, and rendering options. In that sense 3D is more flexible than 2D streaming which does not offer these features and would need to be re-created by the generator of the visualization.
- **Standard visualization methods:** the system should support all graphics primitives needed in scientific visualization, and not be restricted to one or two special cases. The set of primitives in VRML-1 [18] is a good starting point because it is stable and used with good results in many related scenarios. It should be possible to add new types on demand.
- **Integrated option for collaborative working and immersion:** the software should be a suitable core for collaborative virtual environments because most visualization tasks require the collaboration of several peers [20]. Immersion can be achieved by a support of VR hardware, and stereo viewing to ease perception of spatial properties.

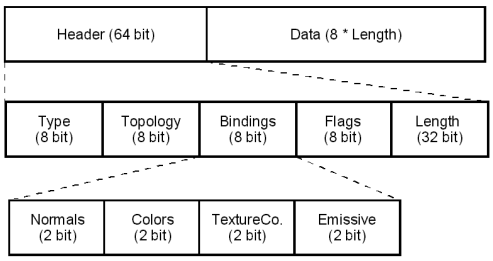

Figure 2: *Structure of a PDU.*

3.3. System Overview

The design follows a client/server architecture (see Fig. 1) that consists of three components:

- generator client
- streaming server
- viewer client

Components exchange protocol commands and visualization data with each other over three communication channels:

From	To	Abbrev.	Channel
generator client	streaming server	G2S	1
streaming server	viewer client	S2C	2
viewer client	streaming server	C2S	2
viewer client	viewer client	C2C	3

C2C extends the software to collaborative virtual environments (CVEs), C2G is used to steer computational simulations.

3.4. The DVR Data Format

DVR is a data format to specify position and transformation of graphical primitives, and other visual properties. DVR is a list of protocol data units (PDU) in binary format, see Figure 2. Fields of a PDU are defined as follows.

1. Type: Identifies one of the following
   - geometry (VRML-1-compliant[18], e.g. FaceSet),
   - material (Phong-shading, texturing, transparency),
   - text element (URL),
   - transformation,
   - camera,
   - light.

2. Topology: sub-class of geometry specified by Type.
3. Bindings: global, vertex-based, or polygon-based binding of additional parameters.
4. Flags: control flags for marshalling data types.
5. Length: number of subsequent PDUs.
6. Subsequent PDUs.

Design of the format has been motivated by the following: the binary representation reduces network traffic, but does not rely on time-consuming compression and decompression. Transformations can be parsed efficiently, and applied more easily by the mapper and the viewer clients, because no stack management is involved. Flexibility for some types of animation is lost, but this is irrelevant when we are interested in dynamic, frequently updated animations that affect large parts of the topology of a scene. Details on the two other formats are omitted in the paper. DVRS is a protocol on top of TCP/IP that describes a sequence of DVR files. DVRP is a bi-directional protocol on top of TCP/IP, used in communication between server and viewer clients. Steering commands such as play, stop, repeat are implemented.

3.5. Implementation of the Visualization Pipeline

The visualization pipeline is implemented as follows.

1. **Filter.** The generator client filters data and processes the result in the second step.
2. **Map.** The generator client maps the set of simulation data to visualization data represented in DVR by a mapping library, LibDVRP, that can be used in a multi-processing environment to build and forward visualization data. The generator client forwards these data to a streaming server. Data transfer to the server is on top of TCP/IP to reduce overhead. TCP is preferred to UDP because it is a reliable protocol. Our approach does not work with lossy schemes. Frame-wise delivery is beneficial for two reasons: it works with any data size, as long as one frame fits into memory. Second, it is very straightforward, and processed by hardware in an efficient way. However, the approach is not optimal if animations are optional, or if the complexity of a single frame exceeds limitations of hardware.
3. **Render/Display.** Remote viewer clients receive the sequence of 3D scenes from the streaming server. With RTSP/DVRP on TCP/IP they steer the simulation. RTSP, the Real-time streaming protocol, is standardized in [5].
4. **Collaboration.** Decoupled from the server, viewer clients exchange control data in real-time over TCP/IP. However, the protocol is experimental and subject to change, see for alternatives [12].

3.6. Generator Client

The Generator must support Fortran, ANSI C, or C++, and optionally MPI to filter and map data with LibDVRP. Other packages could be supported, but we have used standard components for implementation. The API of LibDVRP is still under development, and therefore omitted.

The library is written for DEC OSF/1 and Unix architectures in portable C.

3.6.1. Parallelization of Mapping

With LibDVRP the generator client maps simulation data to visualization data, and forwards the result to the streaming server. This can happen sequentially, or in parallelized mode. The parallelization strategy is up to the application. For example, we have analyzed efficiency of two mapping strategies that work by way of MPI multi-processing. The discussion is based on simplified assumptions and not exhaustive, but makes clear how strategies differ with respect to context.

The first algorithm treats all processes in the same way, except that one is required to forward data to the server while others are idle:

```
Algorithm 1() {
  for each time frame {
    for process n of N in parallel {
      simulate nth partition
      filter nth partition
      map nth partition
        to nth part of visualization}
    merge N parts of visualization
    forward visualization to server}}
```

The second algorithm is an alternative to the first aimed to reduce idle times. The idea is to separate the gatherer from the mappers to balance workload. Note that we introduce an additional process here, hence, measured times cannot be compared to results achieved with the previous algorithm in terms of seconds.

```
Algorithm 2() { // 'split (pipelining)'
  for each time frame {
    do block 1..2 in parallel {
      block 1:  // PE 1..n
       for process n of N in parallel {
         simulate nth partition
         filter nth partition
         map nth partition
           to nth part of visualization}
      block 2: // PE 0
       merge N parts of visualization
       forward visualization to server}}}
```

Algorithms have been implemented, and their run-times and processor allocation measured. The example is taken from a physical simulation (PALM) on a T3E on 32, and 33 processors, respectively. Figure 3 reveals from measurements which task is processed at which moment. Color denotes task, horizontal position and length denote begin, duration, and end of a processing step. The upper half shows patterns created with algorithm 1, the lower half was created with algorithm 2, each for two processors (PE) of which one is the gatherer. The remaining PEs behave similar to PE

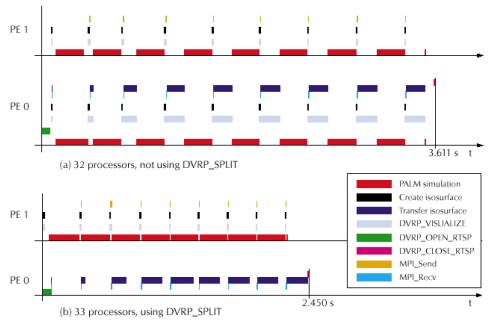

Figure 3: *Processor allocation and workload balancing.*

1 and are omitted. With the second algorithm the user can reduce idle time of each mapping processor at a possible increase of idle time of the gatherer. Second, we can deduce that transfer time in the second algorithm must not exceed time of simulation plus time of visualization (for each processor) because this would mean that the gatherer slows down the visualization process (processes wait for PE 0 to finish transfer). Note that the gathering processor is part of the generator client, and that only the gatherer forwards a 3D scene (i.e. "frame") to the streaming server.

Generator clients like the T3E use an efficient internal network that inter-connects processors. We assume that contention during inter-processor communication is rare, and therefore negligible. Under this premise, which is not necessarily true for clustered systems, a theoretical analysis reveals a formal context of speedup and efficiency based on concepts adapted from [2]. The aim of the analysis is to estimate which algorithm performs better in which case, dependent on network bandwidth, properties of algorithms, and number of processors. Let us assume that N processing units (PE) exist. Let $k = T_s/T_0$. T_s is the time needed by a sequential step that includes simulation and visualization on one PE. T_0 is the time that it takes to transfer a 3D scene (frame) to the server. $T_p(N)$ is the cycle time for simulation, visualization, and transfer of data. Let $S_p(N)$ be Speedup, and $E_p(N)$ denote Efficiency, both defined by [2]:

$$S_p(N) = \frac{T_s + T_0}{T_p(N)} \tag{1}$$

$$E_p(N) = \frac{S_p(N)}{N} \tag{2}$$

For simplicity, we assume linear scalability with an increase of N. Without transporting results to the server, i.e. $T_0 = 0$, we should get maximal Efficiency:

$$T_p(N) = \frac{T_s}{N} \Rightarrow S_p(N) = N \Rightarrow E_p(N) = 1 \tag{3}$$

Use of algorithm 1 leads to:

$$T_p(N) = \frac{T_s}{N} + T_0 \tag{4}$$

$$S_p(N) = \frac{T_s + T_0}{T_s/N + T_0} \Rightarrow S_p(N) = \frac{k+1}{k/N+1} \tag{5}$$

$$E_p(N) = \frac{T_s + T_0}{T_s + NT_0} \Rightarrow E_p(N) = \frac{k+1}{k+N} \tag{6}$$

Overlapping execution by algorithm 2 yields:

$$T_p(N) = max(T_0, T_s/N - 1) \tag{7}$$

If $T_0 \leq T_s/N - 1$ or $k \geq N - 1$ the second term is used, else T_0. The second term is more complex:

$$S_p(N) = \frac{(T_s + T_0)(N-1)}{T_s} \Rightarrow S_p(N) = \frac{(k+1)(N-1)}{k} \tag{8}$$

$$E_p(N) = \frac{(T_s + T_0)(N-1)}{NT_s} \Rightarrow E_p(N) = (1 + 1/k)(1 - 1/N) \tag{9}$$

The same formulas for $T_p(N) = T_0$:

$$S_p(N) = \frac{(T_s + T_0)}{T_0} \Rightarrow S_p(N) = k + 1 \tag{10}$$

$$E_p(N) = \frac{(T_s + T_0)}{NT_0} \Rightarrow E_p(N) = \frac{(k+1)}{N} \tag{11}$$

Calculations are adapted from [10]. Figure 4 illustrates theoretical Speedup and Efficiency as a function of the number of processors for both algorithms and $k = 32$ (arbitrary). The graph illustrates that the splitting strategy is more efficient above a specific point of "break even", i.e. the number of processors for which both algorithms perform equally. The break even point is derived when we set Eq. 5 equal to Eq. 1. Solving for k yields $N(N-1) = k$. Hence, if $k = 32$ then it is true that $N > 6$. For more than 6 processors, algorithm 2 improves performance compared to algorithm 1.

Scalability is higher with algorithm 2, which we can derive from the above formulas by setting Efficiency greater or equal 50%: with algorithm 1 we get $E_p(N) \geq 0.5 \Rightarrow N \leq k + 2$ from Eq. 6, with algorithm 2 $E_p(N) \geq 0.5 \Rightarrow N \leq 2k + 2$ (Eq. 11). With splitting, scalability falls below 50% at a number of processors that is two times higher than in algorithm 1. Smooth data transport can be expected with $k \leq N - 2$.

Algorithm 2 delivers the same results as algorithm 1, but requires half of the data rate: $E_p(N) \geq 0.5 \Rightarrow k \geq N - 2$ for algorithm 1, but in algorithm 2 Efficiency yields $E_p(N) \geq 0.5 \Rightarrow k \geq 1/2(N-2)$. k is proportional to data rate at a constant volume of data because – with respect to the definition of k – T_s is identical in both cases.

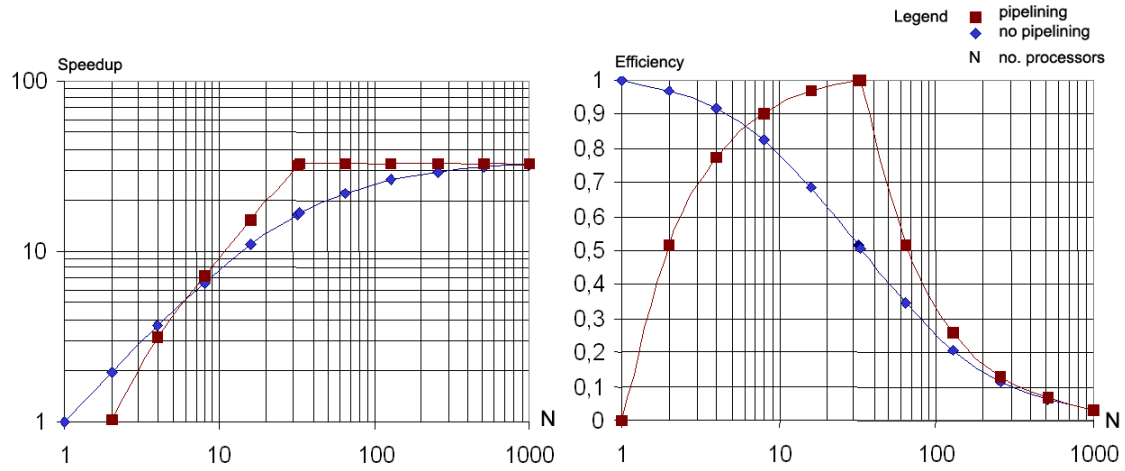

Figure 4: *Speedup and Efficiency graph for $k = 32$.*

3.7. Streaming Server

The streaming server sends DVRS data packages in accordance to steering commands from clients. The server is designed as a component that manages a set of processes. Interprocess communication is implemented through Unix pipes, and for each connected client a process is created to control flow of messages and data. The software allows users to adjust clients to a specific frame rate. In case that not enough frames can be transfered or displayed during a time step, transfer and rendering proceeds with the next frame that should appear at the current clock cycle. By this simple mechanism we guarantee a specified frame rate *on average*. A disadvantage is that playback quality may suffer from an infrequent update of frames on slow clients. As a result, perception on the investigated scene is distorted.

The configuration of the streaming server depends on application-specific requirements. We have used systems with shared memory. However, nothing in the design prevents users to set up the software on clustered machines. The streaming server has been set up on an SGI Onyx 2 Infinite Reality with four processor units R10000 (each with 195 MHz), 2.3 GB main memory, and alternatively on an SGI Origin 200 (two R12000 processors, 270 MHz, 2 GB RAM). A Sun E450 under Solaris is a third option. The server is written for Unix and MS Windows systems.

3.8. Viewer Client

The clients are multi-threaded to process stream retrieval, graphical feedback, and user input in parallel. They retrieve DVR(S) packages by a browser plugin written in C that maps DVR to OpenGL instructions.

Supported collaboration tools include video walls via hardware-supported tele-conferencing, tele-pointers, shared viewpoints, and annotations. Informal tests with up to three clients assess that collaborative working is possible, for instance, 3D entities can remotely be accessed and manipulated between clients (e.g. rotated, translated, synchronized).

Components for tele-immersion include basic support of virtual reality hardware, including stereo viewing. The client can display scenes in mono- or stereoscopic mode. Stereoscopic viewing is possible by use of a stereo splitter with a pair of projectors and polarizing filter glasses. Navigation devices that are compatible to the Logicad 3D Space Mouse are supported.

A multimedia PC comparable to a Pentium 4 with at least 1 GHz clock rate and 512 MB RAM is recommended. Hardware-acceleration of OpenGL is essential and provided by a high-end PC graphics card with at least 64 MB RAM. At least 100 MBit/s Ethernet or a comparable network infrastructure is required. The DocShow-VR client for MS Windows is available at [17]. A basic implementation for Unix and Linux exists. The client is designed as a plugin for browsers that support the Netscape plugin specification.

4. Preliminary Results

4.1. Computational Fluid Dynamics

One viewer client was executed on a Dual Pentium 4 Xeon Personal Computer with 2 GHz clock rate, 1 GB main memory, a 1 GBit/s Ethernet card, and multimedia components. Netscape 6 and Windows 2000 were used. Additional hardware include a H.323 Polyspan Viewstation SP, and a WinTV/PCI grabber card. The 24-bit RGB video texture was

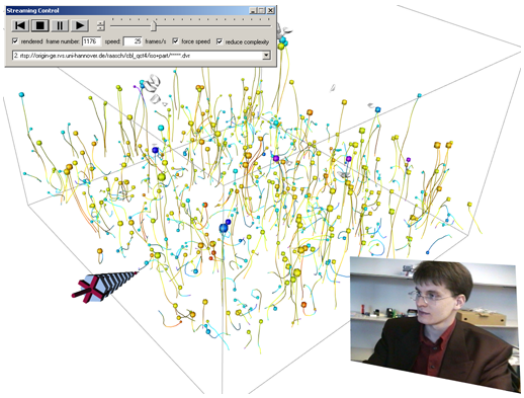

Figure 5: *Visualization of CFD results (simulation of turbulent atmospheric phenomena) using animation of illuminated stream lines. Elements for computer-supported collaborative working: 3D tele-pointer, virtual video wall.*

192 by 144 texels large and updated 25 times/s. We have displayed 3D animations at 25 to 30 frames/s of a pre-stored DVR sequence sent by a Sun E450 in the same building connected over 1 GBit/s Ethernet. The scene was rendered from ca. 43,700 primitives at an image resolution of 1024 by 768 pixels, 24 bit RGB and antialiasing by a graphics adapter from 3DLabs (Wildcat II 5110).

In Figure 5 we can see the flow structure of a convective atmospheric boundary layer. The velocity field is visualized by particles and their trajectories. Particle size is proportional to the magnitude of updraught velocity. Particle color is a measure of the deviation of temperature from the horizontal mean temperature at related particle height (red: warmer than mean, blue: colder than mean). Flow is organized in narrow intense updraughts and broader weaker downdraughts. The Figure was created during a test run based on a pre-created sequence of DVR files with low resolution. More test cases can be found in [9].

What is the upper limit we can realistically achieve by use of the present infrastructure? From measurements we estimate a data rate of at least 0.5 GBit/s on Gigabit/s networks. Smooth animations can be implemented with 25 frames per second, hence one frame must not exceed 2.5 MByte (0.5 Gigabit / 25). For example, if we display an iso-surface that consists of triangles, each with three vertices and one normal (float with 4 Byte), i.e. 48 Byte per triangle, about 50,000 triangles must be rendered (2.5 MByte / 48 Byte) 25 times a second.

4.2. Further Requirements – An Outlook

Another simulation was performed without filtering, mapping, and rendering, compare [15]. It was useful to help us estimating future requirements, i.e. complexity of 3D scenes, that the current system cannot support. Simulation space was 1,000 by 1,000 by 200 grid points, with up to 23 variables (each with 64 Bit) at each grid point. Storing the results of the simulation over thousands of time steps requires management of hundreds of Terabyte, not including visualization data. Roughly estimated, visualization by one iso-surface constructed by means of Marching Cube (without simplification) would generate 2 mio. triangles for one frame. Hence, at 25 frames per second our system would have to transfer and display 50 mio. triangles each second, i.e. up to 2.4 Gigabyte per second because each triangle consumes 48 Byte (as previously described). Albeit these numbers are speculative they show that further optimizations of hard- and software can become necessary.

5. Conclusions

We have presented an application in scientific visualization and computer-supported collaborative working for distributing data in wide area networks with high bit rates. Interactive real-time streaming of immersive 3D scenes is possible. The system does not display voxels, volumes need to be converted to iso-surfaces. Point set rendering is supported, and volume rendering could be added. Future work includes steering, mesh simplification, and session recording. We will complete migration of the client to Linux. The client will support low-cost conferencing systems for collaborative working in simulated immersive environments. Clients will publicly be available.

We must perform more tests and measurements. This is possible in a Gigabit/s network between institutes in Hannover and Berlin. The installation helps federal states to prepare and maintain a shared infrastructure for providing remote access to supercomputers, storage, and visualization facilities (HLRN).

Acknowledgements

Research is conducted at the Lehrgebiet für Rechnernetze und Verteilte Systeme, and the Regionale Rechenzentrum für Niedersachsen, University of Hannover. We gratefully acknowledge support by the Learning Lab Lower Saxony, Swedish Learning Lab / University of Uppsala, and the German Academic Network (DFN). We wish to thank the anonymous reviewers for their valuable comments.

References

1. J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. Law, and M. Papka. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics and Applications*, **21**(4):34–41, 2001.
2. G. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proc. of the AFIPS Conference*, **30**:483–485, 1967.

3. W. Bethel, B. Tierney, J. Lee, D. Gunter, and S. Lau. Using high-speed WANs and Network data caches to enable remote and distributed visualization. *Proceedings of the 2000 conference on Supercomputing*, **2000**:23 p., 2000.

4. K. Engel, O. Sommer, C. Ernst, and T. Ertl. Remote 3D visualization using image-streaming techniques. *International Symposium on Intelligent Multimedia and Distance Education 99 (ISIMADE '99)*, **1999**:91–96, 1999.

5. Internet Engineering Task Force. Real Time Streaming Protocol (RTSP) / RFC 2326. Online at http://www.ietf.org/rfc/rfc2326.txt, accessed on 19. February 2002.

6. Kachina Technologies Inc.. sal: scientific data processing & visualization – software packages. Online at http://sal.kachinatech.com/D/1/index.shtml, accessed on 19. February 2002.

7. C. Law, K. Martin, W. Schroeder, and J. Temkin. A multi-threaded streaming pipeline architecture for large structured data sets. *Proc. of IEEE Visualization 1999 Conference*, **1999**:225–232, 1999.

8. J. Leigh, A. Johnson, M. Brown, D. Sandin, and T. DeFanti. Visualization in teleimmersive environments. *IEEE Computer*, **32**(12):66–73, 1999.

9. S. Olbrich and H. Pralle. A tele-immersive, virtual laboratory approach based on real-time streaming of 3D scene sequences. *ACM Multimedia 2001*, **9**:534–537, 2001.

10. S. Olbrich, A. von Berg, R. Einhorn, S. Heinze, F. Hüsemann, H. Pralle, H. Busch, C. Hege, H. Knipp, A. Merzky, T. Weinkauf, S. Raasch, and K. Meier. Anwendung der Tele-Immersion in Weitverkehrsnetzen: Zwischenbericht M1 22.08.2001. *Technical Report: TK 602 VA 104.1*, RRZN/RVS/IMUK/IFT University of Hannover / Universität der Bundeswehr Hamburg, 25 p., 2001.

11. K. Park, Y. Cho, N. Krishnaprasad, C. Scharver, M. Lewis, J. Leigh, and A. Johnson. CAVERNsoft G2: a toolkit for high performance tele-immersive collaboration. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2000*, **2000**:8–15, 2000.

12. C. Perkins and J. Crowcroft. Notes on the use of RTP for shared workspace applications. *ACM Computer Communication Review*, **30**(2):6 p., 2000.

13. B. Plale, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter. From interactive applications to distributed laboratories. *IEEE Concurrency*, **6**(2):78–89, 1998.

14. C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. *Proc. SIGGRAPH/Eurographics Graphics Hardware Workshop 2000*, **2000**:10 p., 2000.

15. M. Schröter and S. Raasch. PALM – a large-eddy simulation model performing on massively parallel computers. *Meteorologische Zeitschrift*, **2001**(10):363-372, 2001.

16. SGI. SGI – OpenGL VizServer: Home Page. Online at http://www.sgi.com/software/vizserver/, accessed on 5. June 2002.

17. University of Hannover. Tele-Immersion in Weitverkehrsnetzen. Online at http://www.rvs.uni-hannover.de/projekte/tele-immersion/, accessed on 5. June 2002.

18. Web3D Consortium. Web3D Consortium: specifications. Online at http://www.vrml.org/-fs_specifications.htm, accessed on 19. February 2002.

19. WG11(MPEG). MPEG-4 description. Online at http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm, accessed on 19. February 2002.

20. J. Wood, H. Wright, and K. Brodlie. Collaborative visualization. *Proc. of IEEE Visualization 1997 Conference*, **1997**:253–260, 1997.

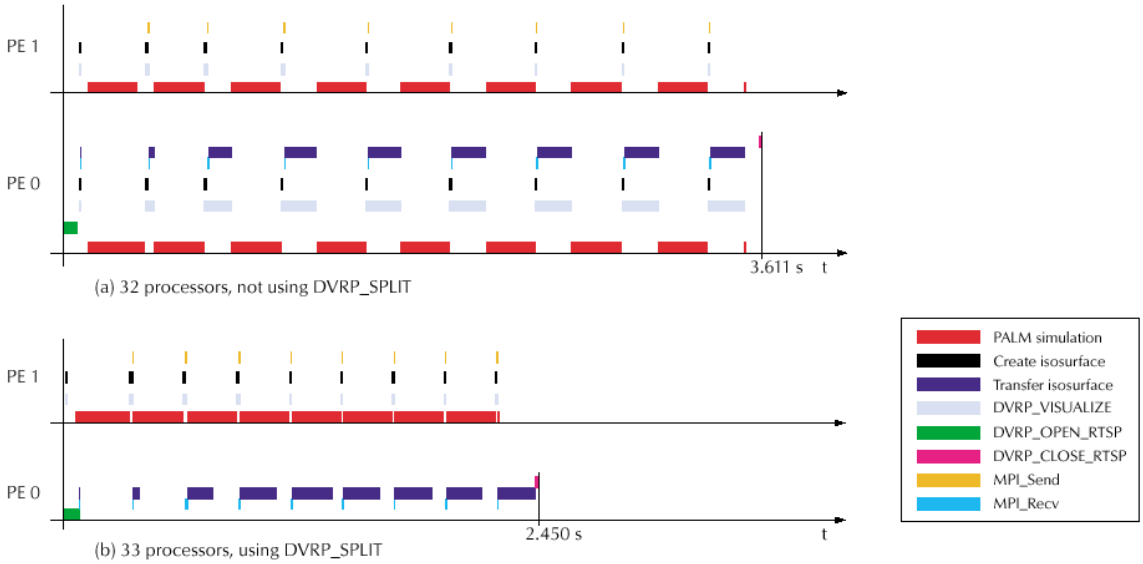

Figure 6: *Processor allocation and workload balancing.*

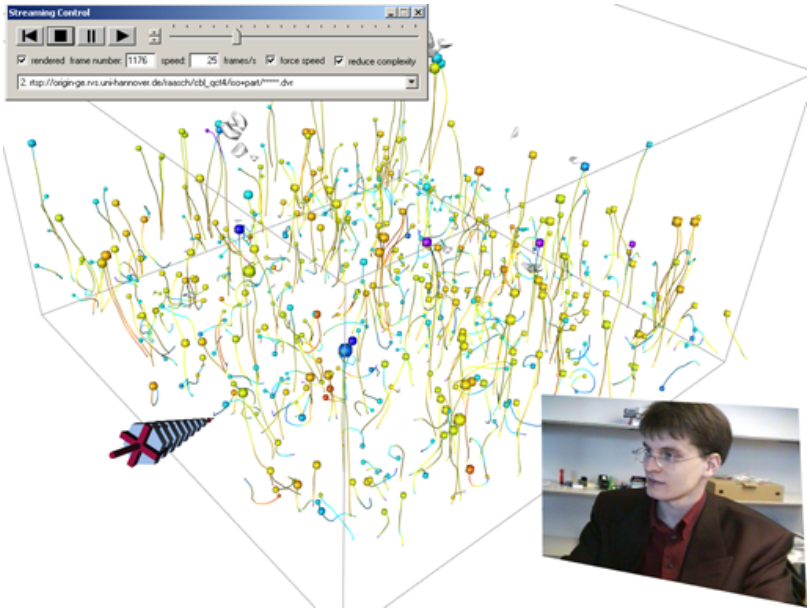

Figure 7: *Visualization of CFD results (simulation of turbulent atmospheric phenomena) using animation of illuminated stream lines. Elements for computer-supported collaborative working: 3D tele-pointer, virtual video wall.*