

Distributed Video Generation on a GPU-Cluster for the Web-Based Analysis of Medical Image Data

Friedemann Rößler ^a, Torsten Wolff ^a, Sabine Iserhardt-Bauer ^{a,b}, Bernd Tomandl ^c,
Peter Hastreiter ^b, and Thomas Ertl ^a

^aInstitute of Visualization and Interactive Systems, University of Stuttgart, Germany;

^bNeurocenter, Department of Neurosurgery, University of Erlangen-Nuremberg, Germany;

^c Department of Neuroradiology, Bremen East Central Hospital, Germany

ABSTRACT

Modern 3D visualization environments for medical image data provide high interactivity and flexibility but depend on the expert knowledge and the experience of the user with respect to the software application. The definition of the visualization parameters is a manual time-consuming process and as a result, inter-patient or inter-study comparisons are extremely difficult. To overcome these drawbacks in case of the analysis and diagnosis of pathologies, standardization of 3D visualization is an important issue. For this purpose automatically generated digital video sequences can be used to convey the most important information contained in the data. In this paper, we present an improvement of our existing web-based service which is now able to calculate the video sequences in much shorter time exploiting the power of a GPU-cluster. The system requires to transfer a medical volume dataset from an arbitrary computer connected via Internet and sends back a number of video files automatically generated with direct volume rendering. To achieve an optimal load balancing of the available resources, the tasks of automatic adjustment of transfer functions, volume rendering, and video encoding are divided into small sub-requests, which are distributed to the different cluster nodes in order to be performed in parallel. An additional preview mode, which renders a number of dedicated frames, provides a direct feedback and quick overview. For the evaluation, we were focusing on the analysis of intracranial aneurysms and were able to show that the system can be successfully applied. Further on, the system was developed in a way that allows easy integration of other analysis tasks.

Keywords: Visualization, Volume Rendering, Standardization, Data Analysis

1. INTRODUCTION

Analyzing medical data is a time consuming and user dependent process. Medical post-processing applications support the physician using different visualization techniques like maximum intensity projection, shaded surface display and direct volume rendering, providing comprehensive insights into complex spatial relationship. Nevertheless, the visualization results are mostly non-reproducible due to user dependent manipulation of the data. Clinical studies, expert exchange, and daily clinical routine require a standardization process in case of medical diagnosing and analysis. In clinical workflows standardized procedures are applied in different fields: For example, radiology departments are using so-called scan protocols which define and guide the image acquisition. Surgeons utilize structured protocols to define all tasks of an operation and the time effort for each task.^{1,2} Using standardized protocols also in case of image analysis and diagnosis is an approach desired by physicians to reduce the complexity and to achieve comparable results.

In a previous publication, we presented a web service,³ which was designed for the standardized analysis of intracranial aneurysms. The standardized process performs all tasks of segmentation, visualization, and documentation of the data for the analysis. The documentation is obtained by automatically generated video sequences of the medical data set. A clinical study⁴ has shown the relevance and the applicability of the web service in clinical routine. Comparisons between manual analysis and the presented approach demonstrated the reliability of the system.

Send correspondence to:

friedemann.roessler@vis.uni-stuttgart.de

A bottleneck of this service, which runs on a SGI Onyx, is the time consuming process of video generation and the fact that the entire analysis of the data consists of several independent videos. In order to overcome these limitations an approach is presented which is based on parallel generation of the videos using a PC-cluster, where each of the cluster nodes is equipped with a modern graphics card (GPU). Thereby, the web service becomes significantly faster and therefore more suitable for clinical routine. In addition to the speedup of the video generation, we integrated a preview functionality, which provides a direct feedback and a quick overview. Furthermore, we developed a standardized description scheme for the videos, which makes the service ready for different medical analysis tasks. In contrast to Muehler et. al,⁵ where the authors present a scripting language for animations, the following paper is focused on the fast generation of the video sequences and preview images.

2. SYSTEM ARCHITECTURE AND WORKFLOW

The video web service system consists of two major components (see Figure 1). On the one hand, there is a web server which provides the web interface and manages the user data. It is based on a Tomcat⁶ HTTP server, an open source project of the Apache Software Foundation, that provides a runtime environment for java servlets and java server pages (JSP). This offers an easy way for the implementation of dynamic HTML pages and the processing of user requests.

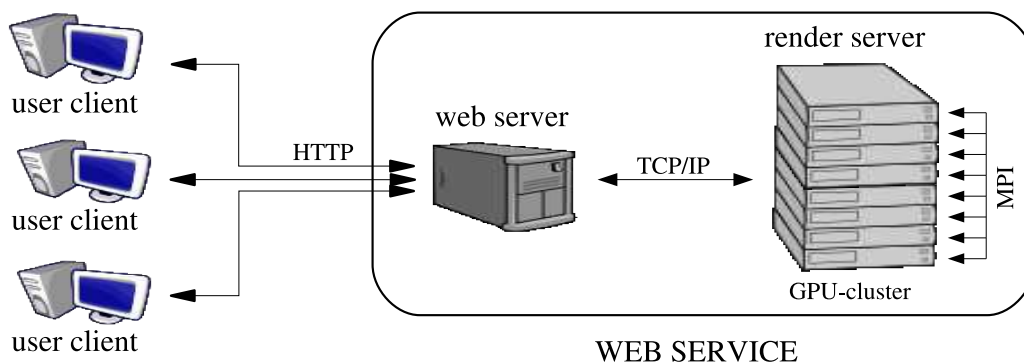


Figure 1. Architecture of the web service composed of a web server and a GPU-Cluster, which builds the render server

The other component is the render server, a GPU-cluster, which performs the rendering and video encoding. This cluster consist of eight standard PCs (4 GB RAM, 2 x AMD Opteron, 2.2 GHz), each containing a NVIDIA GForce 6800 Ultra graphics card with 256 MB RAM. The cluster nodes are connected with a fast Infiniband network. The inter cluster communication is established with the Message Passing Interface (MPI),⁷ a special API for parallel processes. The web server and the cluster communicate via a TCP/IP connection with a standardized protocol for different analysis requests.

A typical workflow of standardized medical analysis performed with our web service is shown in Figure 2. In the beginning a connection to the web server is established via the dynamic web interface. Then, a medical volume dataset with DICOM format is uploaded, some additional rendering parameters are provided and the generation of one or several standardized video sequences is demanded. The web server processes this demand and sends for each video a request to the render server via the TCP/IP connection. At the render server a dedicated cluster node (manager node) takes this request, divides it into several jobs, and distributes them to the other cluster nodes (render nodes). The render nodes execute the jobs in parallel and notify the master node when they have finished. After all jobs have terminated, the render server informs the web server, which then provides the videos to the user via the web interface.

3. DISTRIBUTED VIDEO GENERATION

Among the quality of visualization, the overall processing time of the analysis process is one of the most important criteria for the applicability of the video web service in clinical practice, since a short response time provides a

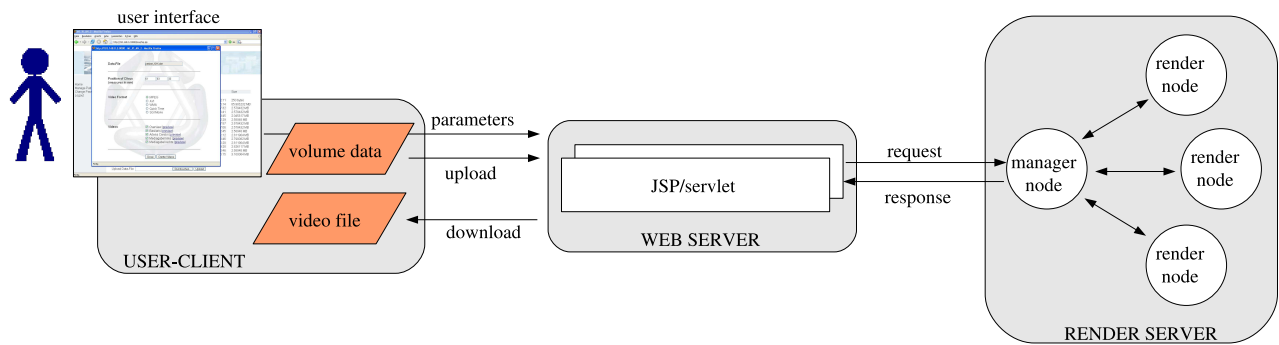


Figure 2. The workflow of video generation: A user requests the video generation via a web interface. The web server takes this request and sends it to the render server. There the request is distributed and performed in parallel. After finishing the resulting videos are provided for download.

fast feedback and permits direct interaction. However, the computation time for rendering and video encoding is restricted to the capabilities of the underlying hardware. The only way to speed up the video generation process on a single machine is to downgrade the quality of the resulting visualization, which is not desired in medical analysis.

An alternative for the acceleration of rendering without loss of quality is the parallelization of the video generation on a cluster computer. Since, depending on the medical problem, several videos of the same data set are needed, an obvious distribution approach is to generate each video on a separate cluster node. Additionally, it is possible to split up a single video into a number of parts, render them in parallel, and merge the results. In the following sections the processing steps of distributed video generation from the initial analysis request to the final videos is described in detail.

3.1. Standardized Analysis Requests

One of the major implementation demands was the applicability to different medical analysis tasks. Furthermore, it should easily be possible to integrate new functionality. To achieve these goals, the web server communicates with the render server via standardized requests, that contain all parameters needed for the control of the analysis result. The data (e.g. the examined volume data set) is exchanged along the shared file system of the web server and the render server.

Currently, the system supports two types of requests, one for the creation of a whole video and the other for the rendering of a sequence of preview images. The required parameters for the video request are the volume file name, the path for camera animation and a transfer function, either explicitly given or by a reference function that is automatically adapted to the actual volume. To visualize only a sub-volume, there are additional parameters to define its center and extent. Furthermore, an optional clip plane can be set. Finally the quality of the output video is controlled by the number of frames per second and the video codec that should be used. A request for the creation of preview images is equally structured, apart from the video parameters. These are replaced by the frames which should be rendered and the target format of the image files.

3.2. Handling a Request

The render server is realized as an MPI program. This means that it acts like a single program consisting of several processes that run on different cluster nodes. The advantage of MPI is that it works with arbitrary hardware. Thus, an MPI program could either run on a cluster, a parallel computer or even on a single PC. If the program is started a predefined number of similar MPI processes are generated and automatically distributed to the available hardware. Since each process is executing the same program, a single MPI process has to distinguish its specific role by its unique rank. Usually, the process with rank 0 is acting as the so-called master, which distributes the tasks to the other processes (the slaves) and merges the results. Figure 3 shows how the work

is divided on the render server. The handling and distribution of render request is carried out by the manager process (master), whilst the render processes (slaves) perform the actual rendering and video encoding.

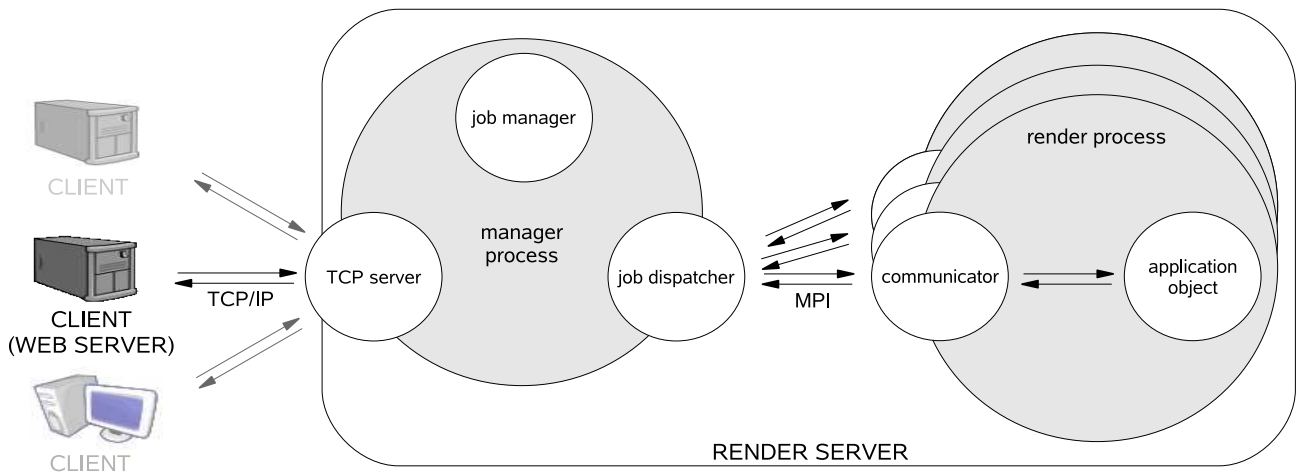


Figure 3. The render server consisting of a single manager process and several render processes, which actually perform the video generation.

The manager process is composed of three major modules: the *TCP server*, the *job manager* and the *job dispatcher*. The TCP server module realizes functionality for external communication. It binds to a specified port and is waiting for incoming connections from different clients. Currently, the web server is the only client that is using the render server, but theoretically any other program (e.g. a stand alone client) could utilize its services, presumed that it has access to the render server's file system. Moreover, several web interfaces, adapted to a specific field of medical analysis, could be provided, applying the same rendering backend.

If the TCP server has established a connection and has received a request, it passes this request to the job manager. That takes the request, generates a number of jobs, which are required for the execution, and stores them in a queue. Finally the job dispatcher distributes these jobs with MPI to the render processes. For this purpose, it permanently monitors the states of the render processes and sends a job to the next one available. This technique guarantees the optimal exploitation of the hardware and permits accepting of multiple requests at the same time, because they are stored until a cluster node is available.

Normally, the jobs are processed in the order they are queued and, if possible, executed in parallel. However, there are cases where a previous job has to be finished before a successor can be deployed. For example the transfer function has to be generated prior to any video sequence or preview image. For this purpose, it is possible to define dependencies on preceding jobs, and a dependent job is put on hold until all predecessors are completed. Furthermore, each job can be given a processing priority to achieve its privileged treatment. This is for example used to ensure that a preview image is served as fast as possible, even if the job queue is crowded with long lasting video generation requests.

3.3. Segmentation, Rendering and Video Encoding

After the generation and distribution of several jobs for a single analysis request, they have to be executed by the render processes, which consist of two modules (see figure 3 on the right). On the one hand, there is the communicator, that corresponds to the job dispatcher at the manager process. It takes an assigned job via MPI, does some preparations and delegates it to the other module, the application object, which has to provide functionality for all tasks, that are needed for automated rendering and video encoding. In Figure 4, which shows the basic video generation pipeline, these tasks are illustrated. At the beginning, the medical volume dataset is loaded to main memory, and, if requested, a sub-volume is built. In the next step, the optimized

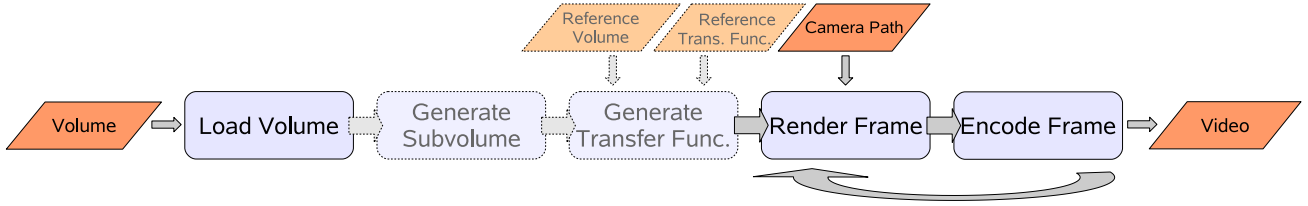


Figure 4. The video generation pipeline from input volume to output video file. The enlightened parts are optional and are only performed, if required.

transfer function is generated and then, the frames along the given camera path are rendered, directly encoded, and stored in the video output file.

The implementation of the application object is based on a generic framework for medical volume visualization, that was originally built for an application for the visualization of functional brain images.⁸ This framework provides a flexible GPU-based volume renderer, where several GPU shader programs for different rendering styles can be applied. For the visualization web service standard direct volume rendering (DVR) with pre-integration⁹ for improved results is used. Certain structures can be segmented and emphasized with a transfer function, which maps the volume’s scalar values to color and opacity. This implicit segmentation is automated with an approach¹⁰ already used in the original web service. The basic idea is, to define an optimal transfer function for a reference data set only once and to automatically adapted it to each new volume. Therefore, the histograms of the two volumes are analyzed and registered with each other in order to obtain a transformation that allows mapping the pre-defined transfer function respectively.

The encoding of the videos is performed on the fly, just after a single frame is rendered. This prevents the time consuming process from saving the uncompressed frames to the file system. For encoding the freely available FFmpeg-library¹¹ is used, which contains a great variety of video codecs and supports a large number of output formats for pictures and video files.

Concerning the parallelization of the video generation, the pipeline shown in figure 4 is not entirely processed but divided into several sub-tasks. There are three types of tasks: those that have to be performed once on each involved cluster node, those that are performed only once on a single node and those that are parallelized and distributed to several nodes. While tasks of the first type can be categorized as preparation tasks, managed by each cluster node on its own, the tasks of the second and third type can be handled directly by the manager process (see section 3.2) and distributed as jobs to the render nodes.

The loading of a volume and the optional generation of a sub-volume are tasks of the first type, because they have to be processed exactly once on each cluster node, even if there are repeated jobs for the same volume. An example for the second type is the transfer function generation, which is expensive and can not be parallelized. Since the resulting table can be stored to the file system and used by other nodes, it has to be performed only once. In contrast to that, the process of rendering and encoding, which belongs to the third type, can be divided into several jobs for generating video parts of arbitrary frame numbers. This can be exploited in order to avoid unnecessary idle times of the cluster nodes. However, it has to be mentioned that the video parts have to be finally merged, which is a potentially time consuming task. Currently, the subdivision of the video requests is only supported for videos of mpeg output format, because there the video parts can be merged by a simple file concatenation.

4. APPLICATION AND RESULTS

We applied our service for distributed video generation to the analysis of intracranial aneurysms, the task which our original approach was already designed for. The standardized process performs all tasks of segmentation, visualization and documentation. The user is only responsible for uploading a DICOM volume dataset. Since aneurysms are often found at the tip of the basilar artery, the left and right cerebral artery and at the communicating artery, the volume is split into up to four sub-volumes, to restrict the visualizations to regions around

critical bifurcations. To define these critical regions the position and extent of each sub-volume is assigned relative to a specific anatomical landmark, the so-called clivus, which has to be provided by the user.

Via the web interface of the service (see figure 5) the generation of up to five videos can be requested, one overview video of the whole volume with an additional diagonal clip plane and four videos of the sub-volumes. After uploading a dataset the system starts the segmentation, rendering and video encoding process described in section 3.3. The automatically adapted transfer function represents the blood vessels as a semi-transparent object with predefined colors to allow a view to objects behind the vessels and to distinguish the boundaries of the vessels themselves. To provide direct visual feedback, an additional preview functionality was integrated, where dedicated frames of the different videos are pre-rendered and presented via the web browser. In addition to the advantage of less rendering time, the download of single images instead of a whole video file is much less time consuming. Our tests have shown, that the process of pre-rendering takes about three seconds. Since this is near real time, the preview could also be used for direct interaction and manipulation of the visualization.

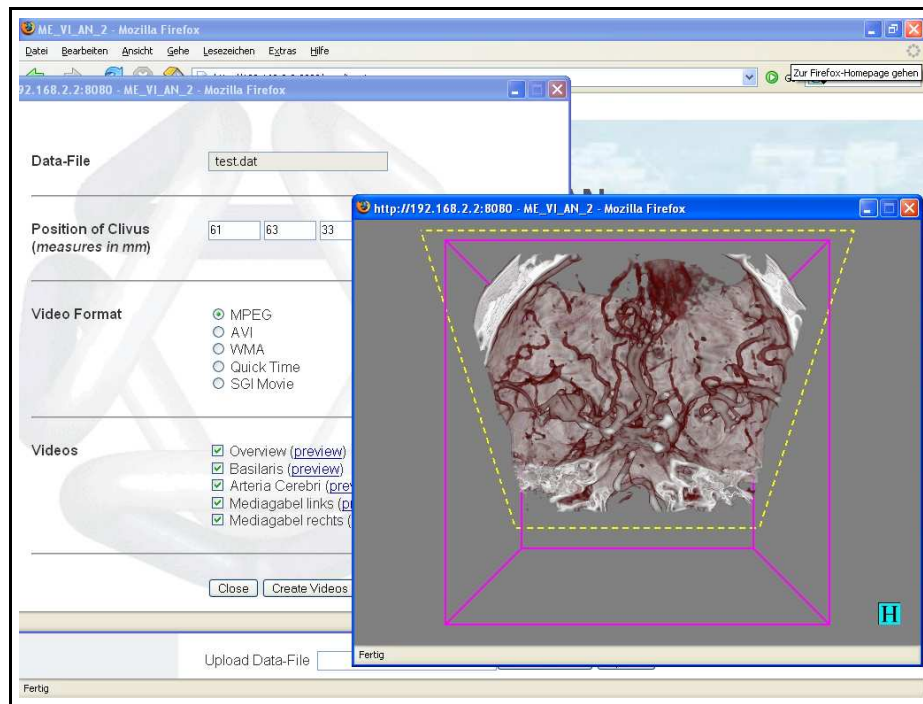


Figure 5. The web interface with a preview image of the overview video including a diagonal clip plane

As stated above the utilized GPU-cluster consist of eight nodes, where one node is serving as manager and seven nodes are actively processing rendering jobs. In order to find the best approach for exploiting the available rendering hardware several parallelization configurations are tested. The tests were carried out with a volume dataset that has an original resolution of $512 \times 512 \times 246$ voxels and a sub-volume size of $256 \times 256 \times 199$ voxels. Each generated video consists of 600 frames and a frame resolution of 640×480 pixels. The videos are encoded in *mpeg*-format. The applied camera path begins with rotations around the coordinate axis and finally zooms in. Figures 7 and 8 are showing some example frames of the resulting video sequences.

Time measurements are taken for three different application scenarios: single generation of the overview video, single generation of a sub-volume video and generation of all five videos in parallel. For each of these scenarios five different subdivision strategies from 600 frames per video part (no subdivision) to 50 frames per video part (twelve parts per video) have been tested. The results are shown in figure 6.

For the simplest distribution approach, each video is rendered as a whole (600 frames) on a separate render node. In this case, the generation of the overview video takes 94 seconds, the generation of the sub-volume video takes 52 seconds. The total generation time for the set of five videos is also 94 seconds, which is equivalent to

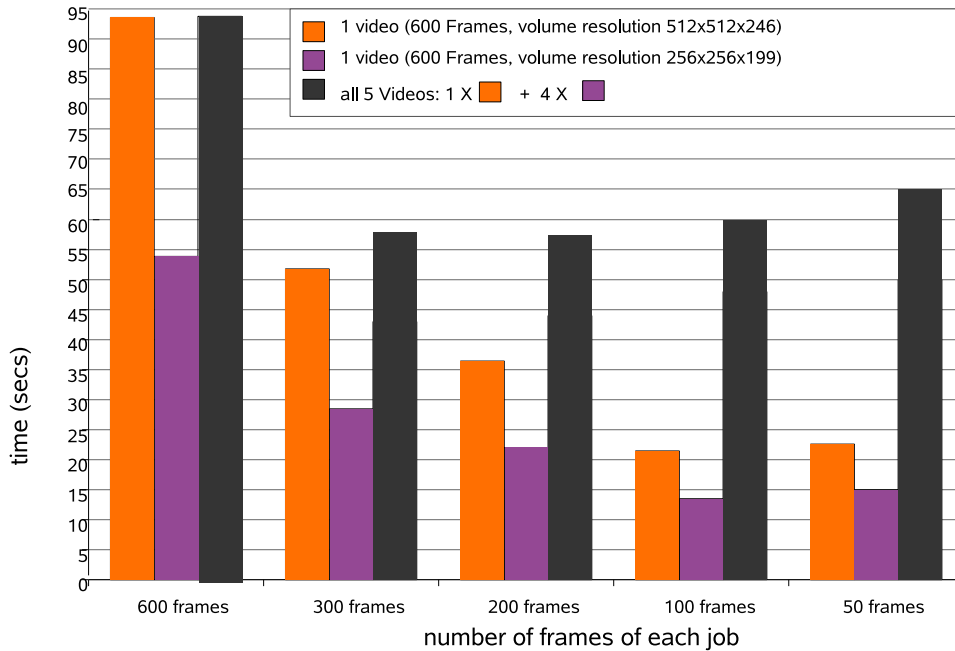


Figure 6. Video generation times for different subdivision strategies from 600 frames per video part to 50 frames per video part. For each case the three bars show the rendering times for the overview video (left), a sub-volume video (middle), and five videos (1×overview and 4×sub-volumes) together (right).

the generation time for the most time consuming overview job. Since there are seven render nodes and only five render jobs, the cluster is not exploited optimally. If the videos are split into parts of 300 frames, there are ten rendering jobs for the five videos, so some of the cluster nodes have to perform two jobs. Due to the better utilization of the hardware the total video generation time goes down to 57 seconds. When the videos are further subdivided there is no further improvement. The rendering times are even getting worse because of the increasing overhead for the distribution and merging of the videos. In contrast to that, the performance for the rendering of a single video is improving down to the splitting of the videos into parts of 100 frames. The reason is, that six jobs are built, which can be performed completely in parallel on the available render nodes. A further split into parts of 50 frames (twelve jobs) brings no advantage. Additionally, it can be noticed, that the generation time for the videos is not decreasing linearly with respect to the number of involved cluster nodes. This is once again due to the overhead of communication. As a result it can be stated, that the hardware is optimally exploited, if the number of distributed jobs is nearby the number of available cluster nodes. However, if the execution time of the rendering jobs differ noticeable, like for the overview video and the sub-volume videos, a subdivision to smaller parts would guarantee a better balanced exploitation of the cluster nodes, whereas the extra effort for distribution management and the assembling of the final videos has also to be taken into account.

Finally, we compared the performance of the video web service with our previous implementation on a single SGI Onyx. There the rendering and video generation time for a similar overview video was about 610 seconds. The same request takes now, with an optimal subdivision of 100 frames per video part, only 23 second, which is a remarkable speed-up of approximately 30 times. For the generation of the set of five videos the improvement is even better.

5. CONCLUSION AND FUTURE WORK

An improved version of a web service for the automatic generation of volume rendered medical video sequences for the standardized medical analysis has been presented. In contrast to our former approach, which was running on a single SGI Onyx, the video generation is carried out in parallel on a modern GPU-cluster. Our results have

shown that this leads to a significant decrease of the overall processing time. Furthermore, the integrated preview mode provides a direct visual feedback. Since the analysis can be performed on any Internet-PC without special hardware requirements the suggested service is perfectly suited for clinical application. A generic design of the service assures that it is not restricted to a certain medical task and easily adaptable to new requirements.

In the future further functionality for enhanced segmentation and registration, for example for the fully automatic localization of the clivus will be integrated. Furthermore, other application areas will be examined.

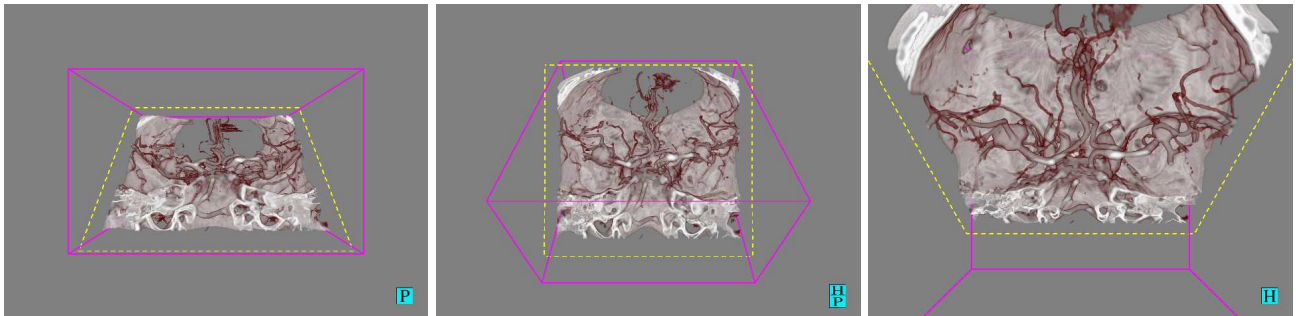


Figure 7. Three example frames of the overview video (start, rotation, zoom) with a clip plane (yellow).

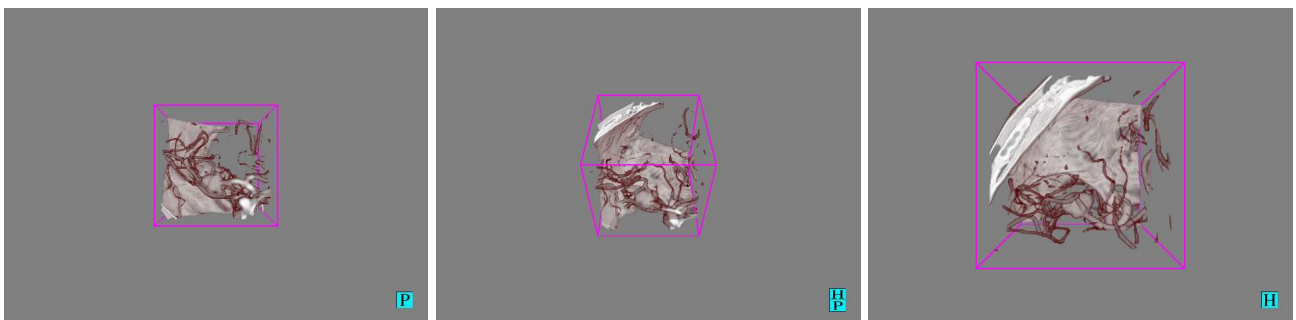


Figure 8. Three example frames of an sub-volume video (start, rotation, zoom) with an aneurysms visible in the middle

REFERENCES

1. T. Neumuth, N. Durstewitz, M. Fischer, G. Strauss, A. Dietz, J. Meixensberger, P. Jannin, K. Cleary, H. U. Lemke, and O. Burgert, "Structured recording of intraoperative surgical workflows," in *Proceedings Medical Imaging 2006: PACS and Imaging Informatics*, S. C. Horii and O. M. Ratib, eds., *Proceedings of the SPIE* **6145**, pp. 54 – 65, 2006.
2. O. Burgert and W. Korb, "ICCAS Research Report 2005," tech. rep., Innovation Center Computer Assisted Surgery (ICCAS), 2005.
3. S. Iserhardt-Bauer, P. Hastreiter, T. Ertl, K. Eberhardt, and B. Tomandl, "Case Study: Medical Web Service For the Automatic 3D Documentation For Neuroradiological Diagnosis," in *Proceedings of IEEE Visualization '01*, pp. 425–428, IEEE, 2001.
4. B. Tomandl, P. Hastreiter, S. Iserhardt-Bauer, N. Köstner, M. Schempershofe, W. Huk, T. Ertl, C. Strauss, and J. Romstock, "Standardized Evaluation of CT Angiography with Remote Generation of 3D Video Sequences for the Detection of Intracranial Aneurysms," *Radiographics* **23**, p. e12, Mar–Apr 2003.
5. K. Muehler, R. Bade, and B. Preim, "Adaptive script based animations for medical education and intervention planning," tech. rep., Otto-von-Guericke Universitt Magdeburg, 2006.
6. Tomcat, "Apache Tomcat." <http://tomcat.apache.org>.
7. MPI, "The Message Passing Interface (MPI) standard." <http://www-unix.mcs.anl.gov/mpi/index.htm>.

8. F. Röbler, E. Tejada, T. Fangmeier, T. Ertl, and M. Knauff, "Gpu-based multi-volume rendering for the visualization of functional brain images," in *Proceedings of SimVis 2006*, pp. 305–318, 2006.
9. K. Engel, M. Kraus, and T. Ertl, "High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading," in *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01, Annual Conference Series*, pp. 9–16, Addison-Wesley Publishing Company, Inc., 2001.
10. C. Rezk-Salama, P. Hastreiter, J. Scherer, and G. Greiner, "Automatic Adjustment of Transfer Functions for 3D Volume Visualization," in *Proceedings of Vision, Modelling and Visualization 2000*, pp. 357–364, 2000.
11. FFmpeg, "FFMPEG Multimedia System." <http://ffmpeg.mplayerhq.hu>.