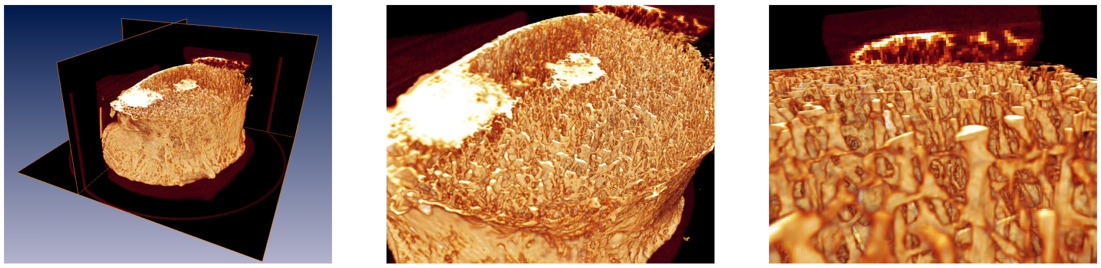
Interactive Exploration of Large Remote Micro-CT Scans

Steffen Prohaska[*] Andrei Hutanu[†] Ralf Kähler[‡] Hans-Christian Hege[§]

Scientific Visualization Dept., Zuse Institute Berlin (ZIB)



Abstract

Datasets of tens of gigabytes are becoming common in computational and experimental science. This development is driven by advances in imaging technology, producing detectors with growing resolutions, as well as availability of cheap processing power and memory capacity in commodity-based computing clusters.

In this article we describe the design of a visualization system that allows scientists to interactively explore large remote data sets in an efficient and flexible way. The system is broadly applicable and currently used by medical scientists conducting an osteoporosis research project. Human vertebral bodies are scanned using a high resolution micro-CT scanner producing scans of roughly 8 GB size each. All participating research groups require access to the centrally stored data. Due to the rich internal bone structure, scientists need to interactively explore the full dataset at coarse levels, as well as visualize subvolumes of interest at the highest resolution.

Our solution is based on HDF5 and GridFTP. When accessing data remotely, the HDF5 data processing pipeline is modified to support efficient retrieval of subvolumes. We reduce the overall latency and optimize throughput by executing high-level operations on the remote side. The GridFTP protocol is used to pass the HDF5 requests to a customized server.

The approach takes full advantage of local graphics hardware for rendering. Interactive visualization is accomplished using a background thread to access the datasets stored in a multi-resolution format. A hierarchical volume renderer provides seamless integration of high resolution details with low resolution overviews.

CR Categories: C.2.4 [Computer Communication Networks]: Distributed Systems—Client/Server I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network Graphics I.3.3 [Computer Graphics]: Picture and Image Generation—Viewing Algorithms J.3 [Computer Applications]: Life and Medical Sciences—Medical Information Systems

Keywords: large data, out-of-core-methods, remote visualization, multiresolution visualization

[*]e-mail: prohaska@zib.de
[†]e-mail: hutanu@zib.de
[‡]e-mail: kaehler@zib.de
[§]e-mail: hege@zib.de

1 Introduction

Datasets of 10-100 gigabyte sizes are becoming more and more common. For experimental data this is mainly due to advances in detector technology, where spatial and temporal resolutions are continuously increasing. If the linear spatial resolution of a detector is increased by a factor s, the resulting size of (uncompressed) spatial data grows by s^3. For time variant data, the temporal resolution potentially increases further. Consequently, the increase of data volumes exceeds the growth of main memory of today's computers. Typical sizes are currently $2048^3 \times 2$ Bytes $=$ 16 GB or even $4096^3 \times 2$ Bytes $=$ 128 GB for a single set of image data. In addition to data acquisition systems, numerical simulations are another important source of large datasets. With the advent of commodity based cluster computing, cheap processing power is becoming widely available, leaving many large datasets in its wake. Data from simulations might be time dependent or even higher dimensional if several external parameters are varied.

These massive data typically cannot be loaded completely into main memory of graphics computers. In designing a practically useful visualization system, one must take this fact into account. Following Law et. al. [22], we see two fundamental but conflicting design goals:

1. The system should be able to process any size of data, on any size computer. Performance should scale well.

2. The system should allow users to quickly identify important regions of the data, and then to enable focused attention to those regions. At any stage of processing the system should remain responsive to user interaction.

External memory algorithms and data structures [37] address the first goal. Cox and Ellsworth [10] discussed these problems focusing on visualization of 3D data. The general goal is to redesign the algorithms to run with minimal performance loss due to out-of-core data storage. The first step is to understand the data access patterns. Then, when possible, the algorithm should be redesigned to maximize data access locality, and to devise a data storage layout consistent with the access pattern, thus amortizing the cost of individual I/O operations over several memory access operations. More recently, cache-oblivious algorithms [15] opened a new way of considering these problems. There, the goal is to optimize algorithms for any kind of memory hierarchy containing caches, without needing to know details of the hierarchy, such as cache or memory sizes.

We focus on the second goal: a system allowing *interactive* exploration of large datasets. Data management becomes a challeng-

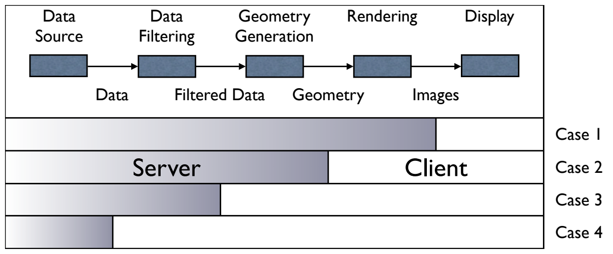


Figure 1: Dataflow in scientific visualization. Case 1: Images streamed from server to client. Case 2: Geometry transferred. Case 3: Data are preprocessed on the server. Case 4: All calculations are performed on the client.

ing task when dealing with huge amounts of data: data consistency, storage capacity, backup facilities, and security each become problems. Therefore data is typically centrally stored and must be accessed *remotely*.

We present a system designed for interactive exploration of remotely stored large 3D image data. The visualization pipeline, as displayed in Fig. 1, can be distributed between client and server in various ways. From the alternative perspectives illustrated in Fig. 1 our system follows case 3, performing data filtering on the server and the rest of processing on the client machine. In our application datasets have a size of roughly 8 GB, but our system also deals well with much larger datasets. A spatially distributed research group now regularly uses our system to access and analyze micro-CT scans of full human vertebral bodies, currently comprising about 0.5 TB of data in total.

The next section will describe this scenario in depth, followed by requirements for the system. After discussing related work, the rest of the paper presents our solution as well as applications of the system. Finally, we draw some conclusions and summarize what can be learned from this work.

2 Scenario

The work presented in this paper is related to bone research conducted in a collaboration of several institutes. Osteoporosis is one of the main targets. According to a NIH definition, this is "a skeletal disorder characterized by compromised bone strength predisposing to an increased risk of fracture. Bone strength reflects the integration of two main features: bone density and bone quality" [27]. The medical researchers are interested in exploring lumbar vertebrae harvested from human cadavers. The main goal is to evaluate structural loss in bone architecture and to gain new quantitative information about the bone metabolism. This will have an impact both on space-flying personnel – bone loss is one of the major problems during long-duration space flights [9] – and on patients with bone diseases on Earth.

Various techniques will be used in the course of the project, including conventional CT and micro-CT [31]; histomorphometric analysis [35], in which the bone is cut into thin slices which are analyzed using microscopy methods; and biomechanical failure load tests. The work described here primarily deals with micro-CT scans of full human vertebral bodies at a resolution of roughly 40μ (computer tomograph type: μCT 80, `www.scanco.ch`). The resulting slices are of size 2048×2048, 2 Bytes per voxel. A complete dataset contains roughly 1000 slices resulting in roughly 8 GB of data. During the project at least 60 of such scans will be acquired, stored on DLT tape and shipped to the research group which is hosting the central data server.

One major goal is to match and compare images from different sources for validation and to facilitate the development of new assessment methods. As an example, a comparison between a photograph and a visualization of the micro-CT data is displayed in Fig. 8. An interesting pending study will be the comparison between micro-CT images taken before and after failure load testing. This new technology provides new insights in the development of micro cracks and the failure of the bone. To support these tasks, the researchers requested the ability to interactively explore the datasets.

3 Requirements

Due to the size of the data, administrative tasks like handling of tapes and backup were decided to be performed at one institution only. Thus, it was decided to store the CT scans centrally in a data repository. The other participating institutions have access to the data via the Internet using 10–100 Mbit/s connections. Simplicity of the data management was considered more important than minimizing the total amount of data to be transferred. Therefore, compression or caching of transferred data is outside the scope of this work.

All users of the project group need to visually inspect the data and to compare them with image data from other imaging devices which are available locally. After identifying subvolumes for detailed analysis, these should be stored locally to allow testing of new quantification algorithms. Stable evaluation procedures must be integrated and run on the whole datasets on a machine connected directly to the data storage.

Budget restrictions demanded a careful investigation whether all the goals could be achieved with commodity based hardware. Local access to high performance visualization systems was not possible for all partners.

Each user should be enabled to explore the data on his own using a graphical user interface. An application environment, users were familiar with, was already present. In this specific case, the workflow was to be smoothly integrated into Amira [33] in order to reduce the training time for the users. A clear migration path for legacy code and legacy file formats as well as import and export of data was also requested.

4 Previous Work

The visualization pipeline (Fig. 1) has five stages: data access, data filtering (selection and modification of the data), generation of graphics data (geometries), rendering (transforming geometries in images), and display. Depending on the distribution scheme between the client and the server(s), various cases can be distinguished (Fig. 1).

Case 1 is an image-streaming approach like VNC [30] or Vizserver [32]. Another solution using a combination of video streaming and remote data access is described in [12]. Approaches that use VNC-like solutions require little or no software development and provide a sufficient solution to visualize remote data for many use cases. But these approaches also have their limitations. Local hardware capabilities are not used, and therefore driving stereo projection or a multi-wall immersive environment like a CAVE can be difficult. Also, running an interactive visualization process on the server holding the data might be restricted, caused by low end or missing graphics hardware, or a batch scheduling system. Loading multiple datasets from different locations in the same visualization session, or storing the data locally for further analysis, is not directly possible. Also, network latency has a direct influence on interactivity and might pose additional problems.

Other image-based techniques are available where the rendering stage is distributed between the client and the server. These can be divided in two classes. The first class of approaches tolerates artifacts in the generated images, thus avoiding intensive computations

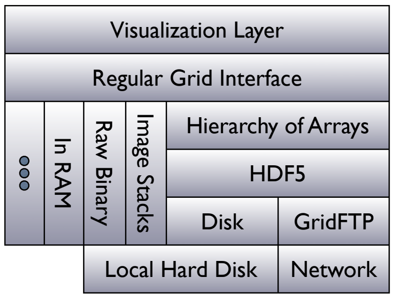


Figure 2: High level design. The *Visualization Layer* accesses data through the *Regular Grid Interface*. Legacy code (*Image Stacks*, *Raw Binary*, *In RAM*) is wrapped to be accessable through this interface. The text describes an implementation using a *Hierarchy of Arrays* stored in *HDF5* which uses virtual file drivers, *Disk* or *GridFTP*, to access the data on the *Local Hard Disk* or via a *Network*. See text for more details.

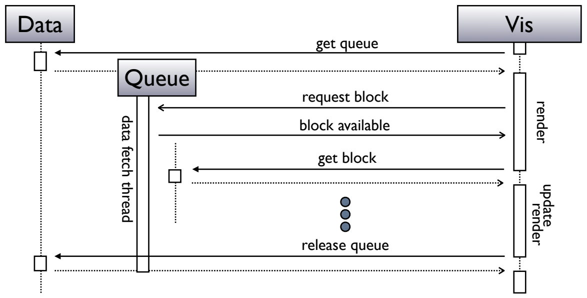


Figure 3: Data access queue sequence diagram. See text for discussion.

or rendering operations on the client side, and reducing the overall expanditure of bandwidth and compute time. High-level image-based remote visualization techniques that have the goal to produce accurate images like the one used in Visapult [5] or the one by Karonis et al. [19] pose requirements in computational power and/or network bandwidth that make them prohibitive for normal users.

Another approach is to separate the filtering and graphic object generation stages from the last two stages (rendering and image display). The server calculates geometries which are transferred to the client which in turn handles the final rendering and the user interaction [18]. Depending on bandwidth, latency and computational power a system might support different distributions of the visualization pipeline [24].

The last category of remote visualization techniques, and the one that fits best to our scenario, places the remote filtering stage on the server side. Because our server is a low-end server dedicated at storing data, parallel octree techniques as proposed by Freitag et al. [14] are not usable. For volume rendering we must generate the octree on the fly on the client machine.

A multi-resolution approach for volume rendering with 3D textures was described by LaMar et al. [21]. They employed an octree based subsampling scheme for uniform scalar datasets to represent regions of the data volume at different levels of resolution. Additionally, they proposed different strategies for view-dependent node selection, and introduced the use of spherical shells as proxy geometries. Weiler et al. [38] presented a similar approach, paying special attention to avoiding interpolation artifacts at the boundaries of adjacent cells at different levels of resolution. Their approach requires that adjacent regions differ by at most one level of resolution. They further improved the technique of opacity corrections to reduce visual artifacts caused by varying sampling distances of the texture slices. Boada et al. [7] presented strategies for adaptive selection of octree nodes from the full pyramidal structure, utilizing data homogeneity and importance criteria. Interactive volume rendering approaches with advanced lightning models using programmable graphics hardware were proposed by Rezk-Salama et al. [29] and Kniss et al. [20]. Guthe et al. [16] presented an interactive texture-based volume rendering scheme for large data sets. They utilize a multi-level approach based on wavelet compression in order to store the whole data in main memory and decompress regions selectively for rendering on-the-fly.

5 Design Overview

This section describes the high level design of our application. We worked to integrate efficient (remote) external data access into an existing environment, taking care to provide compatibility with the original design wherever possible. Our design is based on a stack of interfaces that allow different implementations to be plugged together. See Fig. 2 for an overview.

The highest level of abstraction is the interface to *regular grid data*. It provides the notion of a multidimensional array of data elements. Subblocks at various resolutions can be accessed synchronously or asynchronously. This allows multi-resolution visualization algorithms to progressively retrieve data, as detailed in Sec.7. Memory is managed either by the visualization layer or the implementation of the regular grid interface. Legacy code for accessing large files of raw binary data, stacks of images, or grids residing completely in RAM was wrapped to comply with this interface.

Asynchronous access is achieved by using a queue object (Fig. 3). The visualization layer requests such a queue and sends it non-blocking *request block* messages. Blocks are described by their starting position, the number of voxels, and the stride between voxels in each dimension. Asynchronous access can be implemented using a background thread if an underlying layer does not provide an asynchronous I/O scheme. The background thread fetches data and sends *block available* messages when data becomes available. The visualization layer responds to these messages, interrupting rendering, synchronously retrieving all available data, updating the geometry and finally continuing rendering. After retrieving all required data, the visualization layer releases the queue. At any time it may also kill the queue if the requested data is no longer needed due to user interaction. Note that from the visualization layer's perspective the whole processing runs in a single thread. All thread synchronization problems are completely hidden behind the *queue* interface.

Our implementation is based on a *hierarchy of arrays*. We use *HDF5* [26] as a container for the multidimensional arrays needed by our implementation. HDF5 uses an I/O layer denoted as virtual file drivers. We used this abstraction to implement efficient remote access to the data. Details are given in the next section. An alternative would be the netCDF API [36]. Using MPI-IO [34] to implement this API was proposed by Li et al. [23]. However, HDF5 provides more features than netCDF. The next major release of netCDF (version 4) will use HDF5 as its file format. The chunked data layout and the virtual file driver concept in HDF5 are especially important for our work.

6 External Data Storage, HDF5

HDF5 is a general purpose library and file format for storing scientific data. It provides a container to store multidimensional datasets together with accompanying metadata. The major advantages of HDF5 are the rich set of efficiently implemented features, and the fact that it matured over the years into a stable, wide-

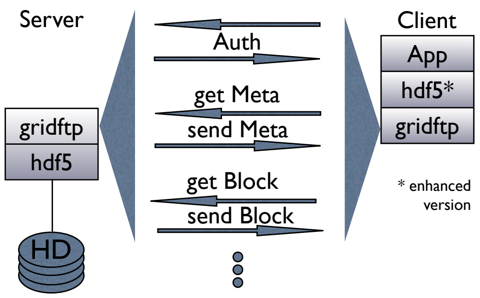


Figure 4: Client server architecture. Client uses enhanced HDF5 as described in the text. Server uses standard HDF5.

spread I/O library and data format. Some feature we rely on are subblock/hyperslab selection; the flexible data model; automatic datatype conversion (endians, types); and some choices in the data layout of the stored dataset. But HDF5 also has its limitations, some of which we are trying to address in our work and are described below. HDF5 is open source which allowed us to change the library internals.

The HDF5 library provides a mechanism for supporting external I/O drivers. This allows the simple integration of various I/O mechanisms in the HDF5 library. In this way, the access to the data uses the same interface independent of the data locality (data stored locally or remotely).

HDF5 natively supports only synchronous I/O. Blocks are read using a single API call which does not return before all data is available. We implemented asynchronous access by placing all read calls to HDF5 into a background thread (described in Sec. 5). This works well for handling complete requests. However interrupting a background thread is still not possible if it is inside a HDF5 call, as this might cause blocking behavior. We split large requests in small blocks to decrease the potential blocking times. Still, the only clean solution would be to add an asynchronous I/O interface to HDF5.

6.1 Data Layout

As mentioned in the introduction, the first step in designing external memory algorithms is to understand the data access patterns. In our scenario, subvolumes will be accessed at various resolutions for rendering and data analysis. A somewhat different use case is slicing. Contiguous storage is optimal for slices made up of the two fastest running indices. The highest possible data locality, one contiguous block of memory, is reached. But slicing in the slowest running dimensions requires scattered access to the whole linear memory range, resulting in dramatically worse performance. Hierarchical indexing schemes [28] or wavelet decomposition could be used to optimize behavior for various use cases. HDF5 provides the possibility to choose chunked storage on a per-dataset basis. In chunked layout the volume is split in equally sized chunks of user

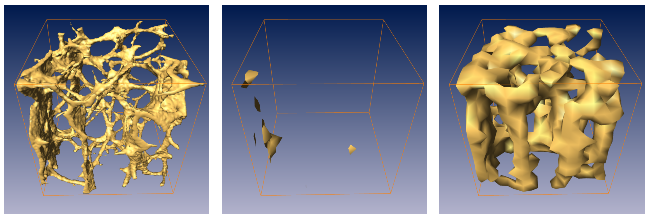


Figure 5: Preview generation. Left: Full resolution. Center: Averaging 8^3 voxels. Right: Maximum filtering 8^3. Isosurfaces with same threshold.

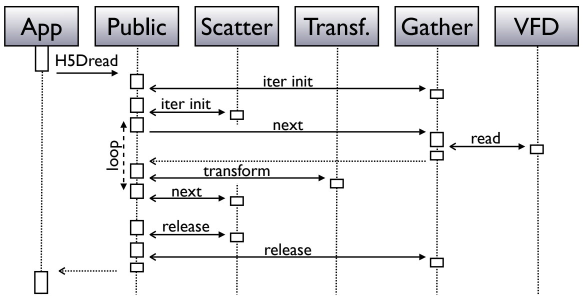


Figure 6: H5Dread, sequence diagram. From left to right: Application, Public HDF5 API, Scatter Iterator, Transformation Filter, Gather Iterator, Virtual File Driver.

specified size, which are stored as contiguous blocks on disk.

We decided to use these HDF5 features and chose a rather simple storage scheme. We store the dataset at the highest resolution using HDF5 chunked layout. Preview versions, which replace 2^l voxels in each direction by one voxel, are added for use when lower resolution data are requested. The bounding box of higher levels is extended to cover the whole volume. The increase of size by the subsampled blocks is roughly 15%. Filters for preview generation can be chosen depending on the data. Simple averaging is not always suitable, as it would cause low-pass filtering, thus hiding high frequency details in the preview. If the original data contains thin structures, it could be better to use a maximum filter during preview generation. Fig. 5 shows a comparison of low resolution versions generated by averaging and using a maximum filter. Previews are regarded as supplementary data, and can be deleted and recreated at any point in time. A wavelet decomposition could avoid the increase in data size, but would increase data management complexity. For keeping the data layout simple and increasing the chance of long term usage (including compatibility issues), we chose to store the original data in the simple layout described.

6.2 Remote Access

To fulfill our application requirements, it was considered sufficient to support only read-only remote access. Write access is only needed locally to store the data at acquisition time.

By analyzing the I/O behavior of our data readers, we observed the reader operations could be separated into metadata reads and other operations. Immediately after a file open, the most numerous operations are metadata reads. The total sizes of the data read by all these operations is usually magnitudes smaller than the actual data. The metadata can be completely transferred to the client machine in a single initial step (Fig. 4). It is stored in the main memory, and used to serve all meta datareads locally.

The main limitation of the original virtual file driver concept is the insufficient support for high-level I/O operations. An I/O driver provides methods for low-level operations like open, close, read and write. Relatively complex operations, e.g. H5Dread, might be broken into a large number of read requests at different positions in a file. However, this is highly inefficient when executed over a high-latency network.

This problem was addressed in [4], and a plan for how to address it in a future version of HDF5 using a virtual data access layer is described in [25]. Our approach was to make a small modification to the HDF5 pipeline that allows the low-level file driver to optimize its operations depending on the initiating operation. Source code is available at `http://www.zib.de/visual/projects/gridlab/hdf5/`.

As illustrated in Fig. 6, the H5Dread operation is composed of three sub-operations. Inside a loop a *gather* iterator reads data from storage, feeding it to the *data type transformation*. In the

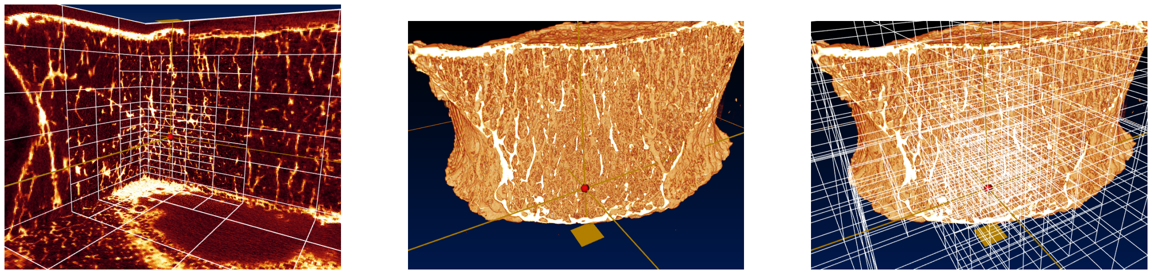


Figure 7: *The left image shows three orthogonal slices through a micro-CT scan. The intersections between the slices and the bounding boxes of the octree nodes are highlighted. The middle image is a multi-level volume rendering of the same dataset. The red point is the center of the region of interest. The right image is a volume rendering from the same view-point that further displays the bounding boxes of the selected octree nodes.*

last step a *scatter* iterator writes the data into memory. When handling remote data, the virtual file driver should know if it serves a H5Dread. Therefore, we decided to implement similar operations for *gather:init*, *gather:next* and *gather:release* on the VFD level. In *init*, the VFD will open a communication channel and initialize the remote operation; *next* will read data from the network; and *release* closes the connection to the remote host. For all other operations which access metadata information, they can be rapidly served using the metadata stored in the main memory which was read from the remote site at file-open time.

We define two operations that are executed on the remote server: one for metadata gathering, and one for dataset read (as described in detail in [17]). Currently the dataset selection specification only supports the mapping of "simple" hyperslabs. However, we plan to extend it to allow selections composed from simple hyperslabs by boolean operations.

After deciding how to modify the HDF5 pipeline, we needed to find a mechanism to use the high-level operations (H5Dread and metadata gathering) on the remote machine holding the data. Naive remote data access techniques that do not support customizable high-level remote operations are not a choice for obvious reasons.

There are a number of representative client-server or remote data access architectures available including the Storage Resource Broker [3], DataCutter [6], GridFTP [2], or OGSA [13]. Each system has its own advantages and disadvantages. For our purposes we chose to base our solution on GridFTP because it provides server-side processing commands, and given its proven performance in terms of large data transfer [1].

7 Rendering

Besides the fact that the raw size of the scans makes it unfeasible to keep the whole dataset in main memory on the available systems, rendering the entire data volume in full resolution would not begin to approach interactive frame rates.

To enable the user to quickly grasp the overall structure of the data, we employ a hierarchical multi-resolution rendering approach. This allows the user to define and change regions of interest and to visualize them at the highest available resolution, while the rest of the data volume is displayed with coarser resolution to maintain interactivity. We choose spherical regions of interest with selectable center and radius parameters. In addition, the user can store subsets of the data (e.g. 2D slices or axis-aligned subvolumes) locally at full resolution for further analysis.

We employ an octree data structure as a multi-level representation of the data volume. The data for the nodes is requested progressively according to the following scheme:

- In a first pass, the tree is traversed starting at the root node. All blocks that intersect the region of interest are scheduled for loading.
- Next, the coarse regions are requested at higher resolution, depending on their distance from the region of interest.

Since the retrieval of the data is performed in a separate thread, the visualization routines are not blocked during the loading phase. If new data blocks have been provided by the reader thread, the visualization modules are notified via a callback mechanism, and the new data is reflected in the next rendered frame of the visualization.

Though it results in a memory overhead of roughly 15 %, we cache the data of coarser octree nodes, even if the data of their subnodes is retrieved. This allows a faster update of the rendering once the user changes the region of interest. Loading further data is stopped once a user-defined upper memory limit is exceeded.

Besides hierarchical visualization modules for orthogonal slicing and the display of height fields, we implemented a 3D texture-based volume rendering module for the octree data structure. In the rendering phase the octree is traversed in a view-consistent (back-to-front) order, starting at the root node. A node is rendered if the data of the subnodes is not available at that point.

In order to further accelerate rendering, the traversal of a subtree can be stopped if the screen space extension of its voxels is smaller than a user-defined threshold (measured in pixel units).

Once a node is selected, a separate 3D texture is defined (or activated if the node was cached from a previous render pass) and rendered utilizing the standard approach for volume rendering with 3D textures [11, 8]. Each node texture is sampled with slices perpendicular to the viewing direction, and blended into the frame buffer.

In order to take advantage of the multi-resolution structure of the data for faster rendering, the sample distance of the slices is adapted with respect to the resolution level of the actual node [38].

8 Timings

The goal of our measurements is to characterize the interactivity and throughput performance of our system. We measure the average time to load one block which quantifies the responsiveness ("user sees something changing on the screen"). The throughput of the system can also be calculated from this number. We measured this time for two different block size settings which influences the system's throughput and responsiveness.

Theoretically, the time to load one data block is *latency* (network, GridFTP, server, client) plus *block size* divided by *network bandwidth*. It is clear that increasing the block size will increase the time to load one block, while simultaneously also reducing the total time to load "all" data blocks. This is because the latency penalty will occur for fewer blocks. In our case, since HDF5 is serializing the I/O requests, we must pay the latency penalty for each block request.

The time to load one block also directly determines the time needed to reach the highest level of resolution around the point of interest. Halving the side length of the blocks reduces their size to an eighth, while adding only one level in the octree. The number of blocks required for loading to reach the highest level is thus only increased by a small fraction (if the octree already had a reasonable height), and the overall time is dominated by the average time to load one block.

Our data server is a dual Xeon 1.7 GHz equipped with a logical volume storage of 170 GByte (15 MByte/sec read rate - hdparm). The implementation of the remote data access server is based on an experimental version of the GridFTP server provided by the Globus Group. We performed all timings using a dataset of size $2048 \times 2048 \times 1024$ with 2 Byte per voxel. The preprocessing step needed to generate the preview levels takes around 25 minutes and increases the size of the file from 8 GByte to approximately 10 GBytes.

Measurements for loading the dataset were performed using two different block size settings and two different network settings. The capacity of the network is always 100 Mbit/s, and the round-trip time in setting 1 is below 0.2 ms (LAN) and 360 ms (WAN) in setting 2. The two block sizes we used, are $128 \times 128 \times 64$ voxels (2 MByte) and $64 \times 64 \times 32$ voxels (256 KByte) respectively. The time to open the dataset (including the filtering and the transport of the metadata) is roughly 6 seconds in all cases.

We measured the time and number of blocks loaded in order to calculate the representative average load time per block. The results for the four different settings are as follows. For 2 MByte blocks: 1.9 s/block (high-latency network) and 0.7 s/block (low-latency network). For 256 KByte blocks: 1.4 s/block (high-latency network) and 0.1 s/block (low-latency network).

Based on these numbers, we can calculate the average throughput of the system (in Mbit/s) as *block size* divided by *average block load time*. Using large blocks (2 MByte) we achieve 23.1 Mbit/s in the low-latency setting and 8.3 Mbit/s in the high-latency setting. For small blocks, the results are 20 Mbit/s and 1.4 Mbit/s for the low/high latency settings.

As expected, in the case of a low-latency network (such as the one of interest for our users) using small data blocks simultaneously results in higher interactivity and high bandwidth. The time to load one block is much shorter than the time needed for a large block (0.1 s vs. 0.7 s), achieving comparable times with respect to overall throughput (20 vs. 23 Mbit/s). In the case of a high-latency network, the large block setting might be a better choice, given the damaging effect of high latency combined with serialized I/O requests. The time needed to load one block is comparable in the two settings (1.9 and 1.4 s), whereas the throughput is much better when using large blocks (8 vs. 1.5 Mbit/s).

In conclusion, a smaller block size gives relatively good responsiveness to the user (taking roughly 3.5 seconds to reach to the full level of detail around the point of interest). We propose a block size of 256 KByte – 512 KByte as the optimal setting for our users connected to the data server over a network with a round-trip time of under 4 ms.

9 Application

The algorithms we have presented were integrated in the Amira visualization environment [33]. Here, we will present a few selected use cases. A major goal is to compare evaluation methods based on different imaging techniques, and to learn about the influence of the micro structure on these methods. Matching and comparing images is a fundamental task to achieve this goal.

In traditional histomorphometry, still the gold standard for analyzing bone structure, the bone is cut into thin slices and analyzed using microscopy methods. Fig. 8 shows a photograph taken of part of the bone side by side with a manually matched subvolume of the micro-CT data. Based on these comparisons, a standard to define a volume of interest for further assessment can be established.

Low resolution CT images are available in clinical practice. Advanced measures for bone quality assessment based on those images would be very helpful in diagnosing and monitoring bone structural changes. These images show the averaged mineral content of the micro structure at low resolution. To learn how well the micro structure can be assessed from the low resolution slices, it is of great interest to locate them in the high resolution data. In Fig. 9 the stages of this process are displayed. Investigating the micro structure might also help to finally build a mathematical model of the deterioration process of trabecular bone.

In the near future, vertebral bodies will be scanned before and after failure load testing. The hope of the scientists is to be able to match locations in the two images. New insight in the development of micro cracks and failure of the integrity of bone will be achievable based on comparisons of the micro-structure before and after load testing. As a result, bone strength prediction might be improved in the future.

10 Results and Future Work

We have presented the design of a visualization application using a distributed visualization pipeline based on remote data filtering. The goal was to provide a tool for interactive exploration of large, remotely stored (medical) data. The major aim was to construct a stable and practically usable system that runs on existing hardware, ranging from PCs to high-end graphics computers, with minimal additional software development.

After studying and comparing carefully the various existing technologies, we decided to choose a combination of established and new components such as HDF5, GridFTP and Amira. The system was designed based on mainly practical requirements. Utilizing existing components helped us to focus on the development of useful visualization techniques for analyzing the remotely stored data. After gathering substantial practical experience, we feel confident that we have found a good solution that is applicable to many other cases. It is possible to build a useful remote visualization system based on existing methods and tools.

The mature interface of HDF5 allowed us to start developing stable software based on local data access while simultaneously enhancing its remote capabilities. We improved the remote performance of HDF5 by making small modifications to its I/O pipeline which reduces the time needed to execute one read operation. We believe that data throughput could be further improved by parallelizing HDF5 operations. In our case, this would be directly achieved by allowing multiple threads to access the HDF5 library in parallel. Also, an important issue is the lack of asynchronous access. In interactive applications, user interaction might render the current read operation unnecessary. Adding support to HDF5 for canceling operations would help handling these cases. Nonetheless, our extended HDF5 version provides efficient remote access – without modifications of the application code.

Our system has been deployed and used in medical research to apply innovative analysis techniques in the study of bone diseases. Due to practical constraints we focused on realized a stable, practically useful working system, at first deliberately leaving aside more advanced concepts that require deeper intervention in the existing code.

Acknowledgments

This study was made possible by grants from the Microgravity Application Program/Biotechnology from the Manned Spaceflight

Program of the European Space Agency (ESA), project AO-99-030, from the European Community (the EC GridLab project, grant IST-2001-32133), Roche Pharmaceuticals, and Siemens AG. We thank the Anatomical Institute at Charité, University Medicine Berlin for provision of the specimens; Wolfgang Gowin and Gisela Beller (Center of Muscle and Bone Research, Charité, University Medicine Berlin) and Jesper S. Thomsen (Institute of Anatomy, University of Aarhus) for handling them. Thomsen also took the photo used in Fig. 8. Markus Burkhart, Bruno Koller and Andres Laib from Scanco Medical acquired the micro-CT scans. We also wish to thank our collaborators and colleagues; special thanks go to Thomas Radke, Andre Merzky and Werner Benger who contributed to the overall ideas of our approach, and to Brygg Ullmer for proofreading. We wish to thank the Globus group, in particular Bill Allcock and John Bresnahan, for their substantial support with the GridFTP server plugin infrastructure and implementation.

REFERENCES

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in highperformance computational grid environments. *Parallel Comput.*, 28(5):749–771, 2002.

[2] W. Allcock, J. Bester, J. Bresnahan, S. Meder, P. Plaszczak, and S. Tuecke. GridFTP: Protocol extensions to FTP for the Grid. *GWD-R (Recommendation)*, April 2003.

[3] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. *Proc. CASCON'98, Toronto, Canada*, 1998.

[4] W. Benger, H.-C. Hege, A. Merzky, T. Radke, and E. Seidel. Efficient distributed file I/O for visualization in grid environments. In B. Engquist, L. Johnsson, M. Hammill, and F. Short, editors, *Simulation and Visualization on the Grid*, volume 13 of *Lect. Notes Comput. Sci. Eng.*, pages 1–6. Springer Verlag, 2000.

[5] 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. In *Supercomputing*. IEEE, 2000.

[6] M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. DataCutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Proc. Mass Storage Systems*, pages 119–133, College Park, MD, March 2000. IEEE.

[7] I. Boada, I. Navazo, and R. Scopigno. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17(5):185–197, 2001.

[8] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In A. Kaufman and W. Krueger, editors, *IEEE VolVis*, pages 91–98, 1994.

[9] J. B. Charles and Critical Path Control Panel. Bioastronautics critical path roadmap. Baseline Document Rev D, NASA, 2003.

[10] M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Visualization*. IEEE, 1997.

[11] T. Cullip and U. Neumann. Accelerating volume reconstruction with 3D texture mapping hardware. Technical Report TR93-027, Department of Computer Science, UNC-Chapel Hill, 1993.

[12] 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 *Visualization*, pages 449–452. IEEE, 2000.

[13] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *Computer*, 35(6):37–46, 2002.

[14] L. A. Freitag and R. M. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *Proc. SC99: High Performance Networking and Computing*, Portland, OR, November 1999. ACM Press and IEEE Computer Society Press.

[15] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms (extended abstract). In *Proc. Symp. Found. Comp. Sci.*, pages 285–397. IEEE, 1999.

[16] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive rendering of large volume data sets. In *IEEE Visualization*, page 53, Boston, 2002.

[17] H.-C. Hege, A. Hutanu, R. Kähler, A. Merzky, T. Radke, E. Seidel, and B. Ullmer. Progressive retrieval and hierarchical visualization of large remote data. In *Proc. Workshop on Adaptive Grid Middleware*, pages 60–72, September 2003.

[18] N. Jensen, S. Olbrich, H. Pralle, and S. Raasch. An efficient system for collaboration in tele-immersive environments. In *Proc. Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 123–131. Eurographics Association, 2002.

[19] N. T. Karonis, M. E. Papka, J. Binns, J. Bresnahan, J. A. Insley, D. Jones, and J. M. Link. High-resolution remote rendering of large datasets in a collaborative environment. *Future Gener. Comput. Syst.*, 19(6):909–917, 2003.

[20] J. Kniss, S. Premoze, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling. In *IEEE Visualization*, pages 109–116. IEEE, 2002.

[21] E. C. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization*, pages 355–362, San Francisco, 1999.

[22] C. Charles Law, William J. Schroeder, Kenneth M. Martin, and Joshua Temkin. A multi-threaded streaming pipeline architecture for large structured data sets. In *Visualization*, pages 225–232. IEEE, 1999.

[23] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A scientific high-performance I/O interface. In *Proc. Supercomputing Conference*, Phoenix, Arizona, November 2003.

[24] E. J. Luke and C. D. Hansen. Semotus visum: a flexible remote visualization framework. In *Visualization*, pages 61–68. IEEE, 2002.

[25] NCSA. HDF5 virtual data access layer. http://hdf.ncsa.uiuc.edu/HDF5/planning/DP/VirtualDataset.html.

[26] NCSA. HDF5 - a new generation of HDF, 2003. http://hdf.ncsa.uiuc.edu/HDF5/.

[27] Osteoporosis prevention, diagnosis, and therapy. *NIH Consensus Statement*, 17(1):1–45, March 2000.

[28] V. Pascucci and R. J. Frank. Hierachical indexing for out-of-core access to multi-resolution data. In *Hierachical and Geometrical Methods in Scientific Visualization*, page 225, 2003.

[29] 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. In *In Proc. SIGGRAPH/Eurographics Graphics Hardware*, pages 109–118, 2000.

[30] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, January/February 1998.

[31] P. Rüegsegger, B. Koller, and R. Müller. A microtomographic system for the nondestructive evaluation of bone architecture. *Calcif. Tissue Int.*, 58:24–29, 1996.

[32] Silicon Graphics, Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043, United States. *OpenGL Vizserver 3.1 White Paper - Application-Transparent Remote Interactive Visualization and Collaboration*, April 2003.

[33] D. Stalling, M. Westerhoff, and H.-C. Hege. Amira - a highly interactive system for visual data analysis. 2004. to appear in: C.R. Johnson and C.D. Hansen (eds.), Visualization Handbook, Academic Press.

[34] R. Thakur, W. Gropp, and E. Lusk. Optimizing noncontiguous accesses in MPI-IO. *Parallel Computing*, 28(1):83–105, 2002.

[35] J. S. Thomsen, E. N. Ebbesen, and Li. Mosekilde. A new method of comprehensive static histomorphometry applied on human lumbar vertebral cancellous bone. *Bone*, 27(1):129–138, 2000.

[36] UNIDATA. netCDF - network common data format, 2004. http://my.unidata.ucar.edu/content/software/netcdf.

[37] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

[38] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *IEEE VolVis*, pages 7–13, 2000.

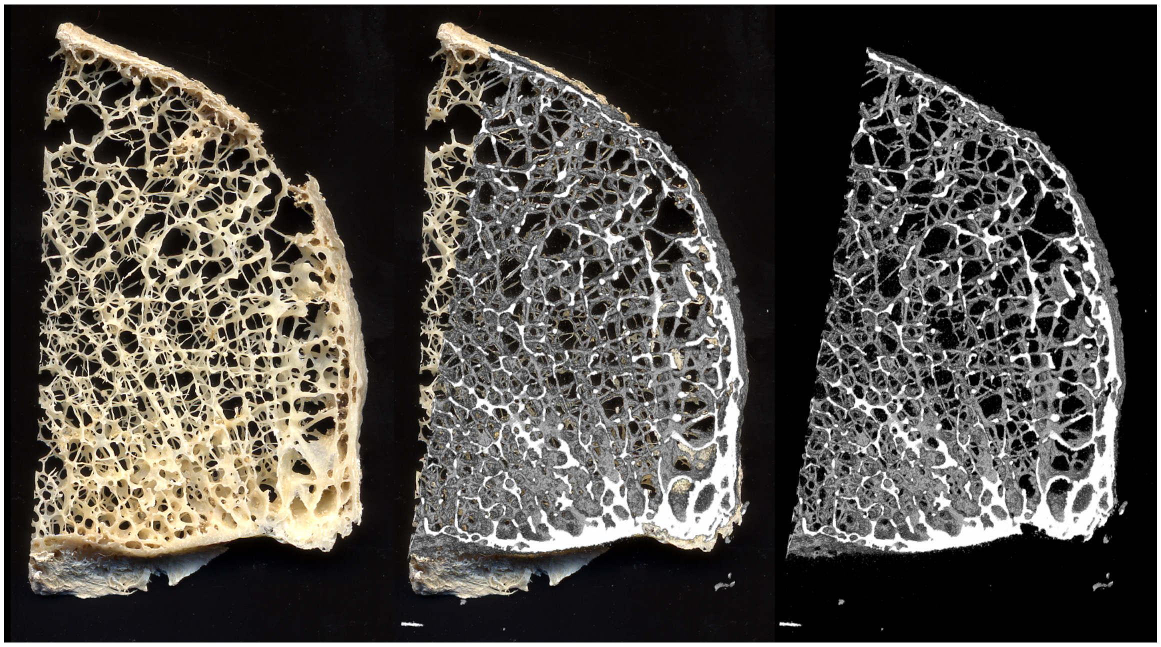


Figure 8: Comparison of visualization with a photograph. Left: Image of a slice cut out of a vertebral body acquired using a flatbed scanner. Right: Volume rendering of a manually matched subvolume (about $700 \times 1000 \times 100$ voxel). Center: Volume rendering in front of photograph.

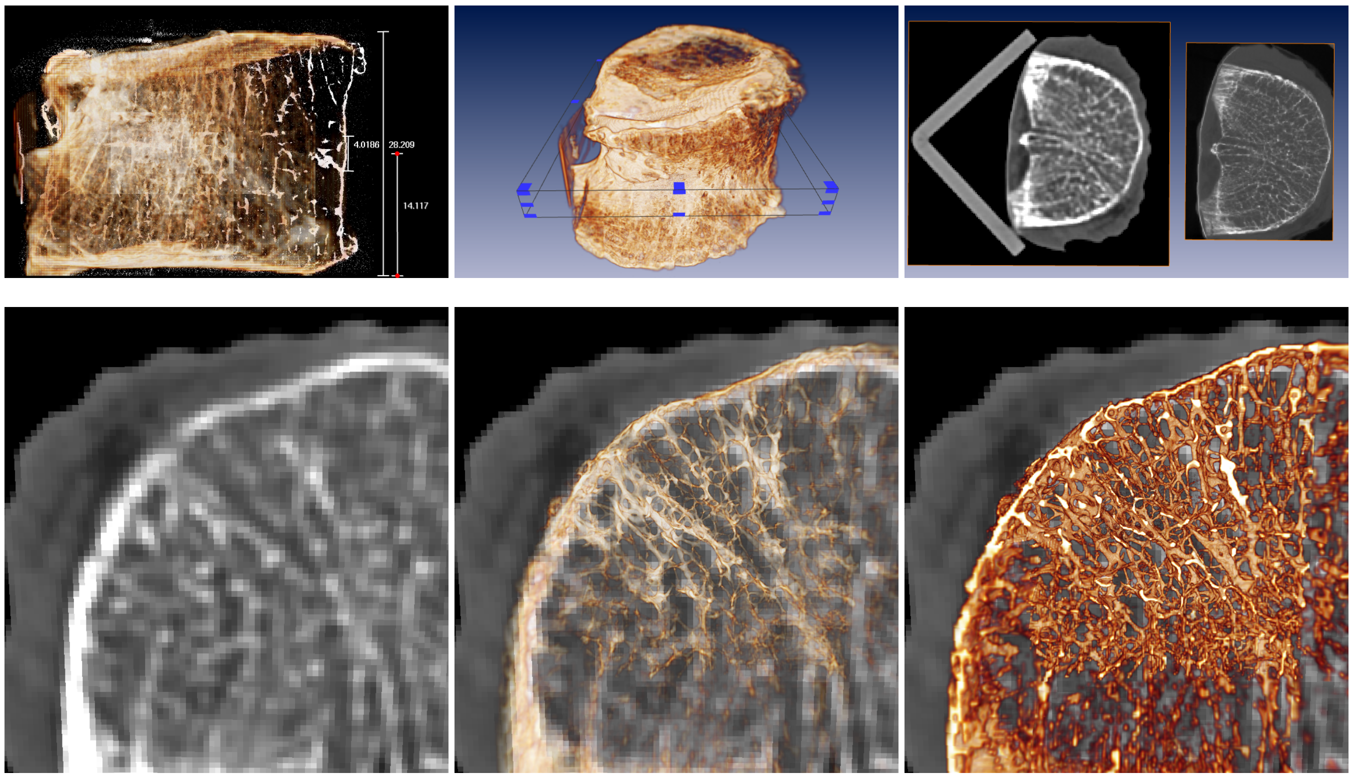


Figure 9: Comparison of image data acquired with a CT scanner for clinical use and data acquired using a micro-CT. Top row: A central 4 mm thick slice is selected and averaged in z direction. The 4 mm slice from the standard CT scanner is visually matched. Left: Orthogonal side view with measurement tools (length in mm). Center: A subvolume is selected using an interactive dragger. Right: The slice from the micro-CT (right) is compared to the slice from the clinical CT scanner. Bottom row: After matching the slice location, the internal bone structure can be directly compared with the clinical CT slice. Left: CT slice. Center: CT slice overlayed with transparent volume rendering. Right: Volume rendering showing the rich trabecular structure – only part of it at full resolution.