Exploring the Connectome – Petascale Volume Visualization of Microscopy Data Streams

Johanna Beyer, Markus Hadwiger, Ali Awami, Won-Ki Jeong, Narayanan Kasthuri, Jeff Lichtman, Hanspeter Pfister

Abstract— Recent advances in high-resolution microscopy allow neuroscientists to acquire volume data of neural tissue of extreme size. However, the tremendous resolution and the high complexity of neural structures present big challenges to storage, processing, and visualization at interactive rates. We present a system for interactive exploration of petascale (petavoxel) volumes resulting from high-throughput electron microscopy data streams. Our system can concurrently handle multiple volumes, and also supports the simultaneous visualization of high-resolution voxel segmentation data. We employ a *visualization-driven* system design that allows us to restrict most computations to a small sub-set of the data. We employ a *multi-resolution virtual memory architecture* for better scalability than previous approaches and handling of incomplete data. We illustrate the real-world use of our system for a mouse cortex volume of one teravoxel in size, where several hundred neurites as well as synapses have been segmented and labeled.

Index Terms—petascale volume exploration, segmented volume data, high-resolution microscopy, high-throughput imaging, neuroscience.

1 Introduction

Reconstructing the human connectome is one of the major scientific endeavors of the 21st century. Connectomics aims to completely reconstruct and map the human brain's neural circuits, comprising billions of neurons and their interconnections, i.e., synapses. By deciphering this network and its properties, scientists hope to gain an understanding of how the brain functions, and how pathologies like Alzheimer's disease or autism develop or can be treated.

However, the immense complexity of the mammalian connectome and the huge amount of imaging data that need to be acquired, stored and—most importantly—processed, present a big challenge for neuroscientists. Finding the connectome of the C. elegans worm, consisting of a mere 300 neurons and their 7000 connections, took over a dozen years to complete [11]. Only recent advances in high-throughput and high-resolution microscopic imaging have made it possible to start tackling the mammalian connectome (e.g., the connectome of a mouse) by acquiring petabytes of volume data with great speed, enabling the accurate reconstruction of detailed neural connections.

Modern microtomes and electron microscopes (EM) can produce volumes of scanned brain tissue with a slice thickness of 25-50 nm and a pixel resolution of 3-5 nm [2], as compared to 200 nm per pixel in optical microscopes. This resolution is necessary in order to be able to trace detailed neural connections at the resolution level of individual synapses. However, with these new acquisition techniques, processing and analyzing the scanned data has become a major bottleneck for connectomics.

For example, an EM volume of 1 mm^3 of brain tissue would already result in a volume of one petabyte in size, and scanning the data with a throughput of roughly 10 Mpixels/s [2] would require an acquisition time of several years. Currently, a high-throughput acquisition process has to continuously stream data over months or even years, which has a huge impact on the way the produced data have to be stored, processed, and visualized. This implies that algorithms for processing and visualization need to be able to handle incomplete data—data that have not been completely scanned yet.

Reconstructing the synaptic connections between neurons is still often achieved by laborious manual segmentation, combined with semi- or fully automatic segmentation approaches [8]. However, interactive 3D visualization of the scanned volume, visual proof-reading of the segmentation, and 3D navigation inside the volume are vital for understanding the data, and must be able to handle large-scale EM data volumes. For example, pre-processing the data into a hierarchical representation as it is usually done for interactive visualization of large volumes incurs an unacceptably large gap between acquisition and visualization. Therefore, it is necessary to develop novel visualization paradigms and systems in order to facilitate the interactive exploration and analysis of large-scale microscopy data streams.

We have developed a flexible volume processing and visualization framework whose design scales to petascale (i.e., petavoxel) volume data, and which is able to deal with incomplete data [6]. In this paper, we motivate the design criteria of our system and describe its integration into the neuroscience workflow for connectomics research, resulting in a large-scale end-to-end system that supports the simultaneous visualization of raw EM data, volumetric segmentation data, and annotations. Furthermore, we will describe the individual components of our system, focusing on how each component is designed to be scalable to petavoxel volumes, including: 1) compact and efficient data storage and retrieval; 2) on-the-fly 3D data construction; 3) interactive 3D visualization of microscopy data streams; 4) integration of volumetric (voxel) segmentation information; and 5) interactive labeling of synapses—the actual connections between individual neurons.

2 Previous Work (Sidebar)

Our system is related to a large collection of prior work, and we only highlight the most important connections here. Seung [11] gives a very good introduction to connectomics and its recent developments, including advances in high-resolution and high-throughput electron microscopy imaging. The work of Bock et al. [2] is an example of how EM circuit reconstruction and the resulting network graph of connected neurons can help in finding a relationship between structure and function of a brain area.

The system described in this paper is in part based on previous work on petascale volume rendering [6] and visualization of neuroscience data sets [7]. Hadwiger et al. [6] introduce a volume rendering scheme for extremely large EM data, focusing on a multi-resolution virtual memory architecture and on-the-fly construction of volume blocks with a thorough performance and scalability analysis, which serves as a basis for the system described in this paper. Jeong et al. [7] describe two systems for interactive exploration and analysis of elec-

- *Johanna Beyer, Markus Hadwiger, and Ali Awami are with King Abdullah University of Science and Technology (KAUST), E-mail: {johanna.beyer, markus.hadwiger, ali.awami}@kaust.edu.sa.*
- *Won-Ki Jeong is with Ulsan National Institute of Science and Technology, E-mail: wkjeong@unist.ac.kr.*
- *Narayanan Kasthuri and Jeff Lichtman are with the Center for Brain Science at Harvard University, E-mail: bobby.kasthuri@gmail.com, jeff@mcb.harvard.edu*
- *Hanspeter Pfister is with the School of Engineering and Applied Sciences at Harvard University, E-mail: pfister@seas.harvard.edu.*

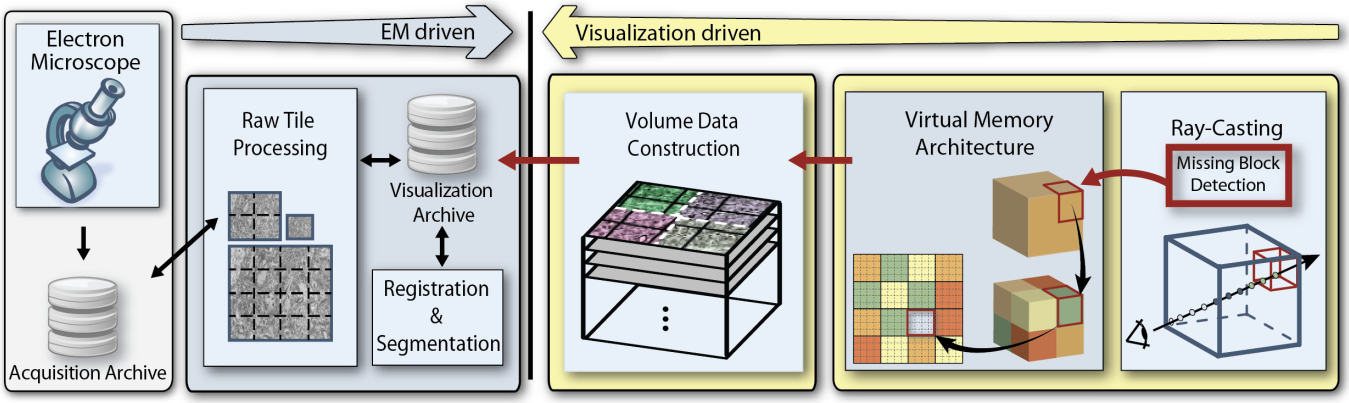

Fig. 1: **System overview.** Petascale volumes are acquired as a stream of image tiles from the microscope. Each raw image tile is processed individually in the input stream. Everything else is *visualization-driven*: Ray-casting operates in *virtual volume space*, detecting missing blocks (*missing block detection*) for visible volume blocks that are not in GPU memory. Only these blocks are then constructed in 3D by stitching and resampling the corresponding tiles from the 2D input stream.

tron microscopy images. Their focus is on manual and semi-automatic tracing of neurons and on-the-fly edge-detection for improved rendering of neural processes. Beyer et al. [1] described a general system for rendering multiple volumes in addition to segmentation data in the context of neurosurgical applications. Although their system also employs out-of-core strategies, it can only handle much smaller data sizes than our system.

Our visualization stage uses GPU volume ray-casting, which has become the most common approach for GPU volume rendering. A main constraint for GPU-based approaches is the limited GPU memory size. In order to accommodate large volumes, out-of-core and multi-resolution volume rendering approaches have been developed, often based on hierarchical octree bricking schemes [10]. These approaches work by partitioning the data into smaller sub-bricks and computing a multi-resolution hierarchy (e.g., octree) of the data in a preprocess. During rendering, only the active working set of these bricks (e.g., all bricks inside the view frustum) have to be downloaded to the GPU, thereby alleviating GPU memory restrictions. However, all previous multi-resolution volume renderers require the multi-resolution hierarchy to be built in a pre-process, which is not feasible for our scenario of dynamically streaming image data. A pre-processing step is also required by all previous systems that support streaming of volume data for progressive rendering, such as the ViSUS system [12]. The *Gigavoxels* [4] and *CERA-TVR* [5] systems perform explicit octree traversal on the GPU by using the kd-restart algorithm. However, this requires holding the entire path from every leaf to the root in GPU memory, and can result in large numbers of updates per frame. Our system avoids many drawbacks of explicit octree traversal by using a virtual memory approach that allows to access any requested resolution directly, without having to traverse the entire hierarchy of coarser resolution levels [6]. Much research has been devoted to volume rendering on large supercomputers [3]. This is especially useful in the context of in-situ visualization of large-scale simulations, where the visualization is computed on the same machine as the data, avoiding the need to move large data. However, this is not a feasible approach for microscopy data. Our data streams do not originate from large-scale simulations, but from acquisition setups that are not directly connected to a supercomputer. Our system streams data to the GPU-based visualization, but only as required by the actual visibility. Such a display-aware approach has also been used before for on-the-fly image alignment and stitching at a resolution that matches the desired output resolution [9].

3 Petascale EM Visualization Framework

Our volume processing and visualization framework consists of two main parts (see Figure 1). The data-driven pipeline, which starts with the actual image acquisition, data storage and 2D mipmap generation, and the visualization-driven part, for visualization and 3D block construction. In the following, we will explain the function of each individual module and their interconnection.

Data generation starts on the left side of Figure 1 and propagates from left to right. In a wider sense, this also includes more complex pre-processing tasks like registration or segmentation. The majority of our system is *visualization-driven* by the actual visibility of small 3D blocks on screen during ray-casting, as displayed on the right side of Figure 1. We operate in *virtual volume space*. This virtual volume space is the reference space of our volume and corresponds to the extents of the 3D tissue block that is being imaged by the electron microscope. If, during ray-casting, the renderer detects that some data block is missing, it issues a request for that data block. This request is handled by the volume construction stage and subsequently the newly constructed block is downloaded into GPU memory. We have paid special attention to a modular design in order to be able to integrate possible future changes such as new data modalities or novel pre-processing algorithms.

3.1 Acquisition Pipeline

Figure 2 depicts our image acquisition pipeline. It starts with taking a tiny sample of a mouse or rat brain and solidifying it using an epoxy resin. The solidified sample is then cut into very thin slices of 25-50 nm using an advanced microtome, Harvard's ATLUM (Automatic Tape-Collecting Lathe Ultramicrotome). To enhance the contrast in the tissue, it is stained with heavy metals. Next, the collected microscope tapes of tissue slices are imaged in a scanning electron microscope with a resolution of 3-5 nm. The microscope acquires image tiles of a fixed resolution (e.g. $12,000 \times 12,000$ pixels) and stores the raw data together with additional meta data in a central *acquisition archive*. The meta data include magnification, position, and orientation of each tile (which we store in an alignment matrix) and current microscope settings.

3.2 Raw Tile Processing

This module is responsible for processing new EM tiles as soon as they arrive from the microscope. This stage works completely automatically. It constantly polls if new tiles have arrived in the acquisition archive, processes them, and stores the data in a compressed form in the *visualization archive*. Figure 2 depicts the raw tile processing stage and the visualization archive in the context of the entire acquisition pipeline. Raw tile processing comprises construction of a 2D mipmap for each tile emitted by the EM followed by a subdivision of each mipmap level into smaller sub-tiles. We chose a sub-tile size of 128×128 for optimized disk access and disk storage. Additionally,

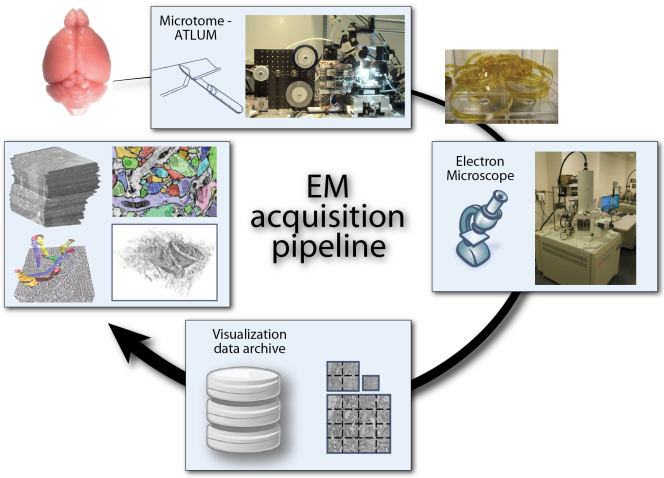

Fig. 2: **Acquisition pipeline.** Tissue samples are cut into ultra-thin slices and imaged using an electron microscope. Acquired image tiles are then stored in a data archive. After 2D mipmap generation, the data can be used for different applications (e.g., visualization, segmentation, fine-grained registration).

smaller sub-tiles can be handled more efficiently in the resampling phase of the volume construction stage (see Section 3.4). Sub-tiles are optionally compressed using JPEG at 2 bpp and stored in the visualization archive.

We store each computed mipmap level of a data set in a separate file. To improve disk access time, we store the sub-tiles in each file in Morton order. This approach preserves data locality and increases cache coherency. The visualization archive also allows external segmentation processes to access the image data and to store segmentation results and any manual labeling of the data.

In principle, the same data archive can be used for the raw and the pre-processed data. However, for organizational reasons it is often better to separate the two archives. The acquisition archive storing the raw microscope data is closely connected to the actual acquisition and can therefore be managed directly by the microscope operators in cooperation with the biologists. All further processing of the data for visualization (or segmentation) is performed in the visualization archive, which is managed by the visualization experts and the biologists and stores the compressed 2D mipmaps.

Since the raw tile processing stage needs to be performed for every new microscope tile, this module must be able to handle the sustained data rate of the microscope, e.g., process more than 10 Mpixels/s at 8 bpp. Currently, our raw tile processing stage achieves a performance of 85 Mpixels/s.

The main motivation for this 2D mipmapping approach (as opposed to 3D mipmap generation) is that it allows us to construct the 2D mipmap of each incoming microscope tile right away, and immediately use it for visualization if it is visible on screen. In contrast, constructing a 3D mipmap would require waiting for all required slices from the microscope in order to compute a 3D multi-resolution hierarchy, or recomputing the 3D hierarchy every time a new slice arrives.

3.3 Registration

Each EM image tile has an affine transformation matrix (i.e., alignment matrix) attached to it which corresponds to the movement of the microscope stage. This matrix is stored only once for each EM tile and inherited by all sub-tiles. The alignment matrix can be iteratively refined by an external registration process to reflect image tile alignment both in 2D and 3D. However, no image data are changed by the registration process, and our raw tile processing stage is completely independent of any registration. Only the alignment matrix is updated by the registration process. Actual stitching of tiles is performed only on-demand in the volume construction stage of our pipeline (see Section 3.4).

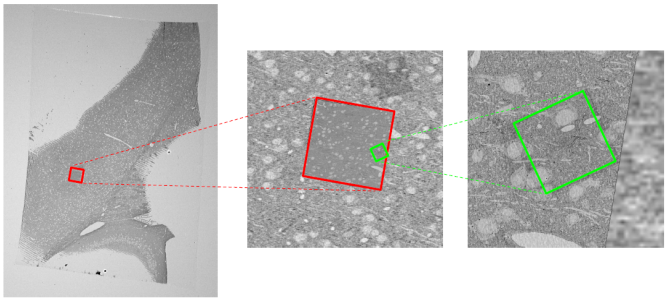

Fig. 3: **On-the-fly registration** of three EM image tiles at different scales.

We can employ an on-the-fly registration technique for *dynamic* EM acquisition. In many cases, the region of interest is much smaller than the entire slice of the original tissue sample. In such a case, without scanning the entire slice at the highest resolution, we can progressively scan the slice at different magnification levels by narrowing down the field of view of the microscope—like zooming into a specific region. Figure 3 shows an example of three EM image tiles at different scales—a low magnification image for the entire view, and two higher magnification images for the region of interest—aligned into a single coordinate system. In this scenario, each EM image tile is acquired at a different image scale and spatial location. To align such images, we use a fixed-size reference grid, e.g., a grid at the screen resolution, and perform the registration of two images on the reference grid. Since the resolution of the images is not the same as that of the reference grid, each image is sub/super-sampled accordingly, based on the magnification level. In our implementation, we use the lower-level image as the background (i.e., reference) image I_L, and the higher-level image I_H is rigidly transformed (i.e., rotation, translation, scaling) to minimize the image difference energy defined as follows:

$$E = \sum_{i=1}^{n} ||I_L(x_i) - (T \cdot I_H)(x_i)||^2, \quad (1)$$

where T is the rigid transformation, x_i is the i-th pixel index on the reference grid, and $I(x_i)$ is the pixel value of the image I at x_i. To minimize E, we use a gradient descent method that iteratively updates the transformation parameters, such as a translation vector, a rotation angle, and a zoom factor, along the negative gradient direction.

The registration process can be done in a semi-automatic fashion if desired: As a new EM image tile comes in from the microscope, the user can interactively navigate a 2D slice view in order to refine the registration if misalignment is visible. The image registration is implemented on the GPU, and its running time is independent of the image tile size because the computation is done on the reference grid. We have observed about 3 ms per single run of registration on a 256×256 reference grid on an NVIDIA Fermi GPU (GTX 580).

3.4 Visualization-Driven Volume Data Construction

The volume construction in our system is entirely driven by the visualization stage (Section 4.1). This means that no data are constructed and loaded into GPU memory if they have not been requested by the ray-caster. This effectively makes the volume construction stage independent of the actual data size. The ray-caster issues a 3D block construction request (at a certain position and resolution level) only if the block is visible on screen and the data request cannot be fulfilled from one of the caches in the visualization stage. Another important feature of our multi-resolution ray-casting scheme is that only the data for the requested resolution level have to be constructed, and that no other resolution levels need to be touched (as opposed to octree approaches).

Figure 4 depicts the volume construction stage. Once a block has been requested by the visualization stage (bottom left in Figure 4), it is

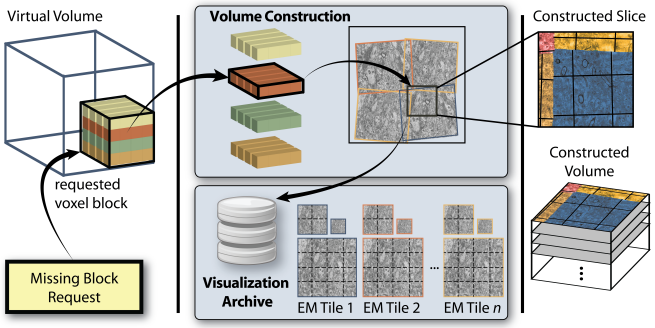

Fig. 4: **Visualization-driven volume block construction.** Only visible 3D blocks in the *virtual multi-resolution volume* are stitched and resampled, computing the result at the requested resolution.

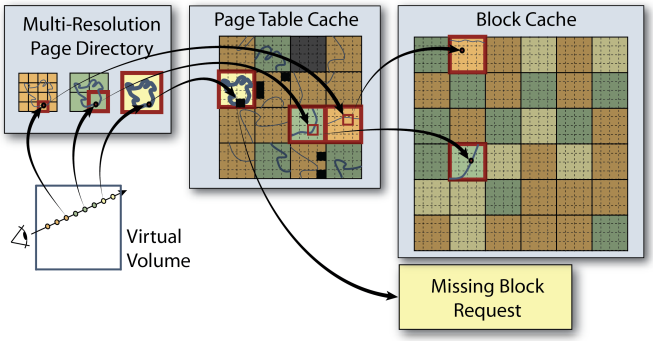

Fig. 6: **Virtual Volume Ray-Casting.** Ray-casting is performed in a *virtual multi-resolution volume*, where each resolution is represented by a *hierarchy of page tables*. Ray-casting accesses actual volume data by performing on-the-fly address translation to access blocks in virtual memory. If a data block is missing, a *missing block request* is generated and propagated to the *visualization-driven volume construction* stage.

constructed in the requested resolution and transmitted to the visualization stage. Block construction consists of two main parts: First, the 2D image sub-tiles that intersect the 3D target block are determined and fetched in the requested resolution from the visualization archive. For efficiently retrieving the correct sub-tiles we have implemented a compact index structure that easily fits into main memory and can still be searched efficiently, based on Morton order traversal of the sub-tiles. The second step consists of stitching and resampling these 2D sub-tiles directly into the 3D target grid. Stitching is determined by the alignment matrix associated with each image tile. We have implemented fast stitching and resampling to any target resolution on the GPU, using texture mapping and fragment shaders. Due to the large slice distance and resulting anisotropy of our EM data (e.g., an aspect ratio of 1:10), we can simplify the 3D block construction process by allowing a 3D target block to be resampled by simply stitching the image sub-tiles in 2D without performing actual 3D filtering, and storing the result into the correct 3D location. As reconstruction filter we can either use GPU bi-linear filtering or higher-order filters implemented in the fragment shader. If the requested data have not yet been scanned by the microscope, we report the block as empty and skip the block construction step.

The modular design of our volume construction stage also allows us to use it as a basis for additional computations, e.g., for automatic segmentation or data analysis, as long as these computations can be performed using 2D or 3D blocks at a certain location with a certain resolution.

3.5 Multi-Threading and System Environment

In order to be able to handle missing data, all our modules are multi-threaded to avoid blocking other computations or delaying the rendering because of uncompleted data requests. The visualization module runs with a separate rendering thread, a GUI or user input thread, and a separate thread for data requests to the volume construction module. Once the renderer has issued a data request, it immediately continues rendering without waiting for the request to complete. The ray-caster is able to deal with incomplete data by either substituting a data block with its lower resolution version, should one be available, or by skipping the block until it has been loaded.

The system environment is based on a client/server network architecture (see Figure 5). Generally, we allow for a flexible setup, where each stage can run on a separate machine, connected via a high-bandwidth LAN. Optionally, the system can be configured to run all modules on the same machine, omitting any network communication. The visualization archive is stored on a shared file system, to allow multiple users to access the data. Rendering is either performed on a separate render server that sends the final images to a thin client, or directly on the PC that displays the final image. All network communication is based on TCP sockets, and can additionally use image compression to reduce network bandwidth.

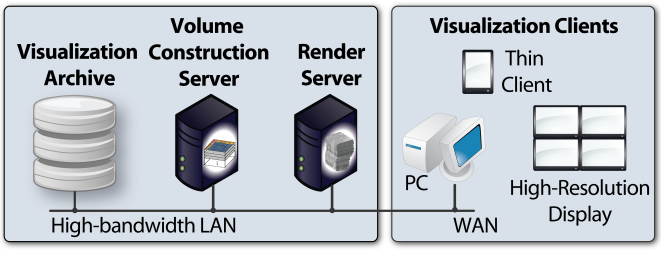

Fig. 5: **System environment.** A configurable client/server setup allows using separate machines for the different stages of our system.

4 Visualization

In this section, we will explain the detailed components of our visualization stage. First, we will focus on our GPU-based volume ray-casting framework [6], based on a multi-resolution virtual memory hierarchy that scales well to extremely large volume sizes (Section 4.1). After introducing our renderer, we will explain the extension of our system to segmentation data, and neuronal connectivity data based on synapses (Section 4.2). Figure 7 shows some renderings of our volume visualization system.

4.1 Volume Rendering

The design of our volume rendering framework differs from previous systems in several important aspects. First, our ray-caster is not based on creating and traversing a tree structure, such as an octree or kD tree. Instead, our design is based on a multi-level, multi-resolution virtual memory architecture that scales well to extremely large volume sizes. This design is more efficient for deep resolution hierarchies as it requires no tree traversal and no tree structure needs to be maintained. Furthermore, it reduces latency by allowing each sample to be fetched directly from any resolution level and enabling switching between resolutions without having to construct intermediate lower resolutions. Finally, it supports arbitrary down-sampling ratios between resolution levels, which enables better accommodation of anisotropic voxel data.

4.1.1 Virtual Memory Architecture

We operate in virtual volume space, which is the reference space that corresponds to the size of the 3D tissue block that is being scanned by the EM. We start by subdividing the volume into small 3D blocks. We use 32^3 blocks, and add a single voxel boundary for correct interpolation between neighboring blocks. Only the working set of these currently required (visible) blocks is resident in GPU memory in a large 3D cache texture that is updated dynamically. To access a sample in

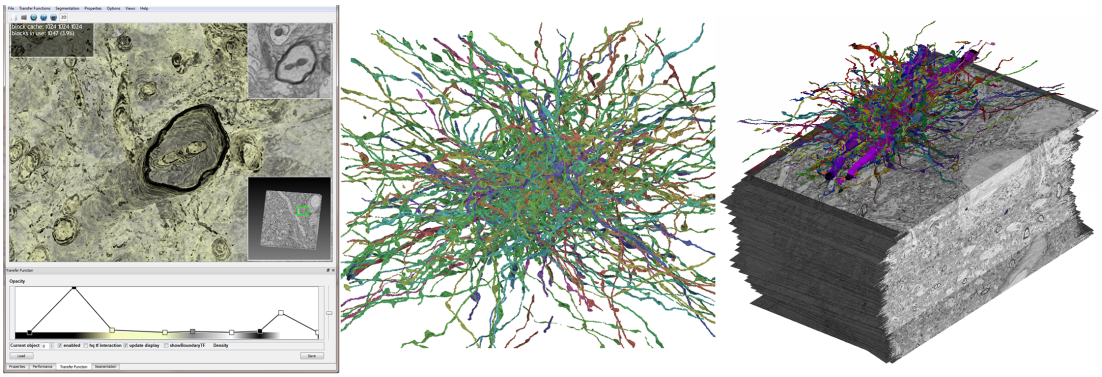

Fig. 7: **Our system** supports the visualization of large-scale electron microscopy volumes, their segmentation information and synaptic connections. Left: Screenshot of our application showing an unsegmented axon. Middle: Segmented axons. Right: Combined rendering of EM data with segmented axons.

the original volume, we now have to translate the sample's position to a coordinate in cache texture space, which is done on-the-fly using page table look-ups. Therefore, the original volume becomes a virtual volume that is accessed via a page table, and only the smaller cache texture and the page table have to be stored on the GPU. If a block is not resident in the cache texture, it is flagged as *unmapped* (i.e., missing) in the page table. However, for very large volumes one indirection layer (i.e., page table) is not sufficient. Therefore, our system not only virtualizes the original volume but can also virtualize page tables.We refer to the top-level page table in the resulting hierarchy as the page directory. Currently, we use two indirection layers, which already enables scalability to several hundred teravoxels [6]. This is in contrast to octree approaches, which require many more levels to be traversed.

For multi-resolution rendering, we conceptually have a separate hierarchy of page tables for each resolution level of the data. However, since the blocks of different resolution levels have the same voxel size (e.g., 32^3), we can map blocks of any resolution level into the same 3D cache texture. The only structure that directly reflects the multi-resolution nature of the data is the multi-resolution page directory.

4.1.2 Ray-Casting Virtual Multi-Resolution Volumes

Ray-casting marches along the ray from sample to sample, performing hierarchical address translation for each sample to map the virtual volume position to the corresponding position in the 3D cache texture (see Figure 6). The sample position on the ray is given by a normalized coordinate in virtual volume space. At each sample point, we compute the desired level of detail (LOD) to use for accessing the corresponding resolution level of the data. We estimate the LOD by computing the projected screen space size of the current voxel. The sample's position and LOD are used for the address translation look-up in the corresponding level of the multi-resolution page directory.

An important property of our ray-casting scheme is that many successive samples along a ray will map to the same page directory and page table entries. Therefore, we can reduce the texture look-up overhead significantly by exploiting spatial coherence and reducing the number of required texture fetches. The closer a page table entry is to the root of the hierarchy (the page directory), the less frequently it needs to be fetched. For example, using 32^3 blocks, for an axis-aligned ray the page table is accessed only every 32 voxels, and the page directory only every 1024 voxels.

Missing data are detected during ray-casting whenever a page directory or page table entry is accessed that does not point to data but is flagged as unmapped. This generates a *missing block request* for the missing 3D block of visible data. These data requests are propagated backwards in the pipeline. If they cannot be met by any of the caches in our system (in GPU or CPU memory, or in the volume construction module), they trigger the visualization-driven construction of volume data from 2D image tiles. To ensure interactive frame rates, we impose a limit on the number of blocks that can be downloaded to the GPU each frame. Furthermore, to decide which blocks are no longer needed and can be swapped out of the cache and discarded we use an LRU scheme and track block usage in the ray-caster.

For further optimization we have implemented empty space skipping on the granularity level of page table entries. If a data block is reported to be empty, it is not downloaded to the GPU and its page table entry is flagged as empty. Our system performs empty space skipping on the EM data by culling against the current transfer function.

4.2 Segmented Data and Synapse Identification

Segmentation plays a crucial role in connectomics research, and is used to partition the data into neuronal structures such as axons and dendrites, to trace which structures are connected by synapses, and to determine their spatial relationships. To this end, our framework supports the visualization of sparse segmentations (where only selected structures are traced), as well as of dense segmentations (where all structures are traced), as well as labeled synapses.

4.2.1 Visualization of Segmented Data

A detailed description of the used segmentation modules and tools is outside the scope of this paper, and we will treat the actual segmentation algorithms as black boxes. However, we assume that the segmentation runs on image data from the visualization archive, and that the same archive is used to store the final segmentation results.

Segmentation data are stored as slices of image data (like the original EM data), where each pixel contains the ID of the labeled object it belongs to. To allow for a large number of distinct objects, we store

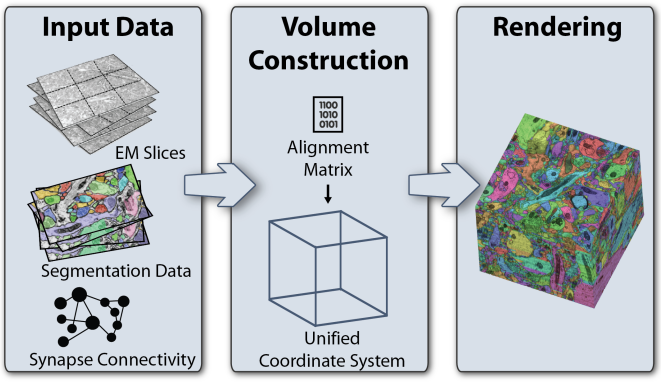

Fig. 8: **Multi-Volume Visualization Scheme.** All input data are stored in the visualization archive. Data requests construct 3D blocks for all requested volumes in a unified coordinate system (i.e., the virtual volume). During ray-casting, multiple volumes can be sampled and combined into the final rendering.

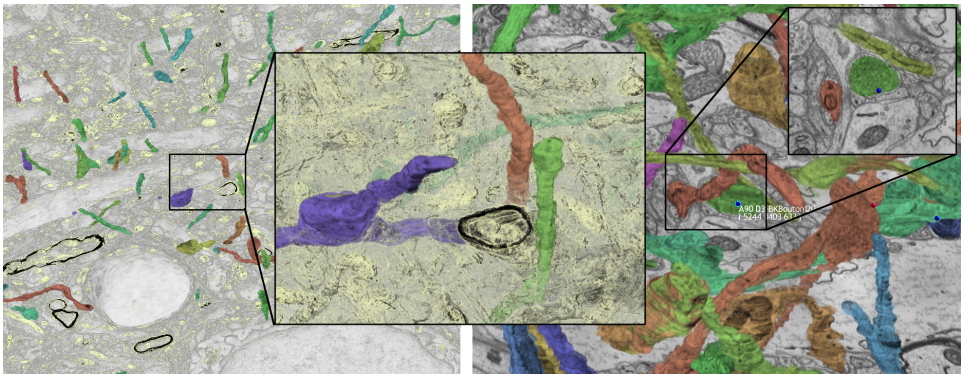

Fig. 9: **Segmented Volume Rendering and Synapse Labeling.** Left and Middle: Different zoom factors and transfer functions for volume rendering segmentation data. The transparent transfer function in the middle image allows to visually follow otherwise occluded structures. Right: Labeled synapse in 3D and slice view. The user can automatically navigate and zoom in to a synapse by selecting it in the 3D view. View parameters and clipping planes are adjusted automatically.

these IDs as 24-bit data, which allows us to store over 16 million different objects. Once the segmentation data arrive at the visualization archive, we compute 2D mipmaps for each slice, as it is done for the EM data (Section 3.2). The main difference is that a different downsampling filter has to be used, because the segmentation data comprise object IDs that must not be interpolated. The straightforward choice for downsampling is to use nearest neighbor filtering, but more elaborate downsampling algorithms can also be used, e.g., using a rank filter.

Figure 8 shows the main steps of our visualization pipeline for segmented data. For highest possible generality, we handle the segmentation volume as an additional data volume and perform multi-volume rendering. This gives us the option to easily extend our system to include additional data volumes, such as functional brain data, in the future. To handle two volumes, we run two instances of the volume construction module, as well as two instances of the virtual memory architecture. We allocate separate cache textures for the EM data and the segmentation data, respectively, because these two types of data are usually of different type, i.e., 8-bit intensity values for EM data, and 24-bit integer IDs for segmentation data.

Actual rendering of the segmentation is performed in the ray-caster. Our system supports different render modes where the object ID of the current sample is used to assign and modify certain properties such as color and opacity, which is then blended with the original EM data. Colors are assigned according to the current sample's object ID.

4.2.2 Synapse Labeling

In addition to the segmentation data, we can also render labeled synapses that are stored in tabular format in the visualization archive. Each entry in the table defines a single synapse consisting of: the position in the virtual volume, a textual label, the IDs of the two objects it connects (i.e., one axon and one dendrite) and some additional meta information.

During rendering we can display all loaded synapses, their labels, and their connections. Additionally, the system allows the user to add new synapses to the data, which are then stored in the visualization archive. Synapses are rendered as small geometric shapes, located at the position specified in the synapse table. To simplify navigation within the volume, the user can select individual synapses which are then automatically centered in the current view. Figure 9, right, shows a volume rendering of segmented axons zoomed-in on a labeled synapse.

5 Results and Discussion

Currently, our collaborating neuroscientists are working on the segmentation and analysis of an electron microscopy data set of a mouse cortex with a resolution of $21,494 \times 25,790 \times 1,850$ voxels, which is a total size of roughly one teravoxel. Over the course of several months, they have segmented several hundred structures by manually tracing them from slice to slice. Most of the segmented structures are axons, which are long and narrow tubular structures that conduct electrical impulses away from the neuron's cell body. The electrical impulses are propagated to neighboring neurons over synapses that connect one neuron's axon to another neuron's dendrite. Dendrites are treelike extensions of a neuron to receive electrical impulses. In our data set, we have segmented 329 axons and 4 dendrites, where each dendrite makes many synapses. The majority of segmented axons are oriented along the z direction, on average spanning over 490 slices of the data set, with about a dozen of them spanning over the entire 1,850 slices, and the smallest spanning only over a couple of slices. In our data set a segmented axon consists on average of over 6.2 million voxels, with a minimum of twelve voxels and a maximum of 37.5 million voxels. The segmented axons constitute less than 0.2% of the entire data set. We have detailed information on 263 synapses, including their location, label, and IDs of the axon and dendrite it connects. On average, in our data set each segmented dendrite is connected to 53 labeled synapses, with a minimum of one and a maximum of 101 synapses. Axons, on the other hand, on average only have two labeled synapses, with a minimum of one, and a maximum of seven synapses.

Figure 7 and Figure 9 show different renderings of this data set, including the segmented axons and labeled synapses.

5.1 Performance

We have tested the performance of our system on three 12-core dual-CPU 3 GHz machines with 48 GB CPU RAM, and NVIDIA Quadro

Table 1: **Performance numbers** for the different stages in our system. For the raw image processing stage, the number indicates the number of megapixels that can be processed per second for 8-bit pixel data. For the registration and the stitching and resampling (i.e., block construction) stages, the numbers indicate the number of megapixels the stage can output per second. Transfer function 1 (TF1) is a linear ramp for color and opacity, transfer function 2 (TF 2) is a more transparent transfer function, as used in Figure 9, middle. The viewport for volume rendering was set to 1024×768.

module	performance	
raw image processing	85 Mpixels / sec	
registration	20 Mpixels / sec	
stitching & resampling	30-65 Mpixels / sec	
	TF 1	TF 2
EM volume rendering	75 fps	12 fps
segmented volume rendering	70 fps	9 fps

6000 GPUs with 6 GB GPU RAM. Our system is implemented in C++, the ray-caster uses GLSL, and for the tile processing we use CUDA and OpenMP. In our current setup, these three machines run the raw tile processing, the volume construction, and the visualization stage including the user interface, respectively. This setup allows us to exploit parallelism between the different stages and to maximize the available cache sizes. All network communication is done using TCP/IP and Winsocks2 over a 1 Gb network. Table 1 shows timing results for the 3D block construction and the raw image tile processing, as well as frame rates for ray-casting the one teravoxel data set, including segmented volume rendering. For measuring the ray-casting frame rate we have used two different transfer functions, the first one is a linear ramp, the second one is a more transparent transfer function allowing to see farther inside the volume.

5.2 Scalability Discussion

This section discusses the major scalability aspects of our system: scalability of our volume representation including multi-volumes, volume traversal and ray-casting.

With our design we have created a system that is scalable to rendering petavoxel data. Our virtual multi-resolution volume representation is extremely scalable due to the small number of hierarchy levels that are needed for the page table hierarchy. Two or three levels are sufficient for extremely large volumes, resulting in easily manageable page directory sizes [6]. In our current implementation, we use two page table hierarchy levels (with a voxel block size of 32^3), which, for example, allows rendering a 4 teravoxel volume with a page directory size of only $32 \times 32 \times 4$. This enables easily accommodating multiple volumes and handling multiple page directories, as well as their corresponding page table hierarchy. Even a data set of several hundred petavoxels could be represented by a page table hierarchy with three levels, with page directory sizes under 64^3 [6].

Volume traversal in our system is extremely efficient. To access an arbitrary resolution level, we only have to traverse the very compact page table hierarchy (i.e., two or three levels), effectively resulting in an $\mathcal{O}(1)$ traversal time for accessing any resolution level. In contrast, octree-based schemes have to traverse the tree from the root to the requested resolution level, which is logarithmic in the number of voxels in the octree. Especially when looking at high resolutions, which is very common in the typical neuroscience use-case, these differences show up clearly in practice. This is illustrated by a more detailed scalability analysis in [6].

For multi-volume rendering and segmented volume rendering we have a separate page table hierarchy for every volume. This has the advantage that very sparse segmentation data can be stored in a smaller cache than very dense EM data. To reduce the number of required textures and texture fetches, one can potentially share page directory and page table look-ups among multiple volumes. However, this requires the same size and layout in all cache textures, which in turn can lead to inefficient cache usage.

Our system assumes that the current working set (i.e., all visible data of the desired resolution) fit into the block cache in GPU memory. If the current working set does not fit into the block cache, our system can lower the requested resolution. Another option would be to perform multiple rendering passes on a single GPU [5], or to utilize parallel rendering on multiple GPUs.

5.3 Discussion

The main objective of our system is to enable exploration of petascale EM volumes in 3D at interactive frame rates. A main advantage of our design is that it allows the frame rate to be completely decoupled from the time it takes until missing data have been constructed and downloaded into the GPU cache textures. Therefore, our rendering system never stalls because it is waiting for new data. Naturally, this approach incurs a *latency* until all visible data have arrived in the requested resolution and a fully complete image is visible on screen. This is very similar to the latency encountered in Google maps, for example, but with 3D data blocks instead of 2D map tiles. The overall latency varies significantly, and ultimately depends on the number of new 3D blocks that must be constructed for a new frame in addition to already cached data. However, in a typical scenario that number is often small, leading to low latencies.

6 Conclusions

We have presented a scalable system design for the interactive 3D exploration and navigation of segmented high-resolution EM data. We have illustrated the major design choices of our system: *visualization-driven* volume data construction, and a novel *multi-resolution virtual memory* scheme. Segmentation data can be integrated by following a multi-volume data handling and rendering approach that scales well, even for multiple volumes. Scalability to petascale (petavoxel) EM data streams is achieved by (1) decoupling data acquisition from the multi-resolution hierarchy required for visualization (this is made possible by the visualization-driven approach of volume data construction), and (2) decoupling the resolution hierarchy of the data from the hierarchy for volume sampling during ray-casting (this is made possible by our virtual memory scheme).

In the future, we want to develop intuitive 3D navigation metaphors for large-scale volume data. We also want to extend our system to support distributed volume rendering, especially for handling and rendering multiple volumes that are too large to handle efficiently in a single GPU out-of-core memory approach.

References

[1] J. Beyer, M. Hadwiger, S. Wolfsberger, and K. Bühler. High-quality multimodal volume rendering for preoperative planning of neurosurgical interventions. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Visualization 2007)*, 13(6):1696–1703, 2007.

[2] D. Bock, W.-C. Lee, A. Kerlin, M. Andermann, G. Hood, A. Wetzel, S. Yurgenson, E. Soucy, H. S. Kim, and R. C. Reid. Network anatomy and in vivo physiology of visual cortical neurons. *Nature*, 471(7337):177–182, 2011.

[3] H. Childs, M. Duchaineau, and K.-L. Ma. A scalable, hybrid scheme for volume rendering massive data sets. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 153–162, 2006.

[4] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of 2009 Symposium on Interactive 3D Graphics and Games*, pages 15–22, 2009.

[5] K. Engel. CERA-TVR: A framework for interactive high-quality teravoxel volume visualization on standard PCs. In *Posters at Large-Data Analysis and Visualization 2011*, 2011.

[6] M. Hadwiger, J. Beyer, W.-K. Jeong, and H. Pfister. Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Scientific Visualization 2012)*, 18(12):2285–2294, 2012.

[7] W.-K. Jeong, J. Beyer, M. Hadwiger, R. Blue, C. Law, A. Vasquez, C. Reid, J. Lichtman, and H. Pfister. Ssecrett and neurotrace: Interactive visualization and analysis tools for large-scale neuroscience datasets. *IEEE Computer Graphics and Applications*, 30(3):58–70, 2010.

[8] W.-K. Jeong, J. Beyer, M. Hadwiger, A. Vazquez, H. Pfister, and R. Whitaker. Scalable and interactive segmentation and visualization of neural processes in EM datasets. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE Visualization '09)*, 15(6):1505–1514, 2009.

[9] W.-K. Jeong, J. Schneider, S. G. Turney, B. E. Faulkner-Jones, D. Meyer, R. Westermann, C. Reid, J. Lichtman, and H. Pfister. Interactive histology of large-scale biomedical image stacks. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1386–1395, 2010.

[10] E. LaMar, B. Hamann, and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proc. of IEEE Visualization*, pages 355–362, 1999.

[11] S. Seung. *Connectome: How the Brain's Wiring Makes Us Who We Are*. Houghton Mifflin Harcourt, Feb. 2012.

[12] B. Summa, G. Scorzelli, M. Jiang, P.-T. Bremer, and V. Pascucci. Interactive editing of massive imagery made simple: Turning atlanta into atlantis. *ACM Transactions on Graphics*, 30(2):7:1–7:13, Apr. 2011.