SSECRETT and NeuroTrace: Interactive Visualization and Analysis Tools for Large-Scale Neuroscience Datasets

Won-Ki Jeong, Johanna Beyer, Markus Hadwiger, Rusty Blue,
Charles Law, Amelio Vazquez, Clay Reid, Jeff Lichtman, Hanspeter Pfister

Abstract—Recent advances in optical and electron microscopy allow scientists to acquire extremely high-resolution images for neuroscience research. Datasets imaged with modern electron microscopes can range between tens of terabytes to about one petabyte in size. These large data sizes and the high complexity of the underlying neural structures make it very challenging to handle the data at reasonably interactive rates. To provide neuroscientists flexible and interactive tools for their scientific work we introduce SSECRETT and NeuroTrace, two systems that were designed for interactive exploration and analysis of large-scale optical and electron microscope images to reconstruct complex neural circuits of the mammalian nervous system.

Index Terms—Neuroscience, connectome, segmentation, volume rendering, implicit surface rendering, graphics hardware.

1 Introduction

Understanding the function of the human brain is one of the most challenging research areas in science due to the extremely complex underlying structures. For example, an adult human brain has more than 100 billion neurons and 250 trillion neural connections. In addition, the size of neuronal cells can be extremely small, such as thin dendritic spines of 50 nanometers in diameter or synaptic clefts of 20 nanometers in size. Connectomics [11] is a field of research to develop methods, from data acquisition to analysis, to reconstruct such complex and detailed neural connection maps of a nervous system. In connectomics research, high-resolution imaging plays a central role in order to identify and resolve nanometer-scale neuronal cells. Optical and electron microscopes are the imaging techniques commonly used to acquire high-resolution images in neuroscience. A typical optical microscope can attain resolutions of 200 nanometers per pixel, while a modern electron microscope (EM) can attain resolutions of up to three to five nanometers per pixel.

Although high-resolution imaging has opened the door to the reconstruction of detailed neural connections, this new technology also poses challenging problems in handling and processing large-scale datasets that are not well addressed with existing methods. For example, with a resolution of five nanometers in the x-y plane and 30 nanometers of slice thickness, the EM scan of a tiny tissue sample of 1 mm^3 of brain tissue is about one petabyte of raw data. One immediate problem is how to store and retrieve such huge datasets efficiently. Automated scanning devices, such as the ATLUM developed at Harvard University, can produce raw image data at a rate of about 11 gigabytes per second. The data storage should be able to provide a reliable mechanism to store streams of raw data and to allow the access of the data in arbitrary location with a minimal latency.

Another challenging problem is how to process and manipulate such large datasets to extract scientifically meaningful information in a more compact form with reasonable processing time. This data processing includes image filtering, segmentation, and visualization. In particular, processing large-scale EM datasets poses a very challenging problem because many existing techniques that work well on relatively low-resolution data modalities, such as CT and MRI, just will not work on high-resolution datasets. For example, existing vascular segmentation techniques work reasonably well for axon segmentation on optical microscopy images, but they may not be directly applicable to feature-rich, high-resolution EM images. Similarly, many deformable image registration methods developed for CT and MRI may not work well on large stacks of optical or EM data alignment.

In the past, only little attention has been paid to tackling large scale biomedical image processing problems. Many well-known image processing tools and libraries are not capable of handling extremely large datasets at interactive rates. Most existing tools and algorithms require the entire input data to fit into the main memory for processing. In addition, the current common practice for segmentation in optical and electron microscopy is a time-consuming and laborious manual process. This manual process becomes a major bottleneck in the entire workflow as data sizes increase.

Therefore, to cope with ever increasing data sizes and to provide users with more flexible and interactive workflows for neuroscience research, we have developed two software tools, SSECRETT (Serial SECtion REconstruction and Tracing Tool) and NeuroTrace [3]. SSECRETT is a 2D slice-based volume exploration and manual axon tracing tool for extremely large-scale neuroscience datasets. It is based on a client-server architecture where the dataset resides on the server side and the client can request an arbitrary 2D cross-section view of the dataset using a socket connection. Our simple solution follows the standard multi-resolution approach of dynamically loading data on demand. Since only a small amount of data can be viewed at any one time, we only load data as it is needed.

On the other hand, NeuroTrace specifically focuses on interactive segmentation and high-quality 3D visualization of high-resolution electron microscope datasets. The system combines local 2D level set segmentation with 3D tracking to extract the 3D tubular structure of neural processes without the burden of processing the entire input dataset. Our visualization method is based on an on-demand local image filtering and adaptive volume rendering technique to achieve interactive performance. NeuroTrace employs an out-of-core multi-resolution octree data structure for large-scale volume rendering and a batch processing framework for off-line segmentation of multiple neural processes. We also utilize the massively parallel processing capability of modern graphics processing units (GPUs) to accelerate many time-consuming and computationally demanding tasks.

The proposed two separate tools, SSECRETT and NeuroTrace, fit nicely into the unified, semi-automatic neural process segmentation

- *Won-Ki Jeong, Amelio Vazquez, and Hanspeter Pfister are with the School of Engineering and Applied Sciences at Harvard University, E-mail: {wkjeong, amelio, pfister}@seas.harvard.edu.*
- *Johanna Beyer is with the VRVis Research Center, Vienna, Austria, E-mail: johanna.beyer@vrvis.at.*
- *Markus Hadwiger is with King Abdullah University of Science and Technology (KAUST), E-mail: markus.hadwiger@kaust.edu.sa.*
- *Rusty Blue and Charles Law are with Kitware, Inc., E-mail:{charles.law,rusty.blue}@kitware.com*
- *Clay Reid and Jeff Lichtman are with the Center for Brain Science at Harvard University, E-mail: jeff@mcb.harvard.edu, clay_reid@hms.harvard.edu*

and visualization workflow. SSECRETT provides an intuitive and interactive method for fast inspection of large input data without requiring high computing power on the client side. This is the first stage of the workflow for the user to find the region of interest quickly with minimal user time and effort. Once the region of interest has been selected, NeuroTrace can extract the detailed 3D geometry of neural process and visualize it using high-quality 3D rendering techniques.

2 PREVIOUS WORK

The most widely used neural image processing tool is *Reconstruct* [1]. This tool has a simple 2D slice viewer with basic paintbrush editing functions to manually draw boundaries of neuronal cells in both optical and electron microscope images. Lu et al. [7] extend Reconstruct with a region-growing segmentation method for semi-automatic segmentation. Some other open-source and commercial software packages, such as the NeuronJ plug-in for ImageJ (http://rsbweb.nih.gov/ij/), Imaris from Bitplane (http://www.bitplane.com/) and Amira (http://www.amiravis.com/) provide automatic segmentation and tracking functions. However, none of them has the ability to handle arbitrary large data size with complex 3D neural structures and to support data retrieval over the network as SSECRETT does. Unlike optical microscopy, only a handful of research efforts have been made for processing electron microscope images. Jurrus et al. [4] and Macke et al. [8] proposed automated slice-to-slice segmentation algorithms using deformable models on electron microscope images. However, they did not discuss the scalability of their methods to large-scale datasets, and their methods are only applicable to a certain type of electron microscope images such as serial block-face scanning electron microscopy (SBFSEM).

Traditional work in scalable volume visualization and processing has focused mainly on multi-resolution approaches and parallel volume rendering on clusters. LaMar et al. [6] were one of the first to introduce a hierarchical (octree) bricking scheme for GPU-based volume rendering. Recently, Gobetti et al. [2] introduced single-pass octree raycasting on the GPU. At each frame they encode a compact octree indexing structure into a small texture, and perform stackless octree ray-casting. Müller et al. [10] introduced a GPU-based cluster system for volume rendering of large data. The system employs a sort-last rendering scheme, where the volume is first subdivided into a kd-tree and distributed to the cluster nodes, which perform GPU-based ray-casting on the individual bricks. However, most previous approaches only try to deal with data in the gigabyte range, which is several orders of magnitude lower than our data requirements.

3 SSECRETT

As data grows to petabyte sizes, it becomes difficult or impossible to copy or move the data in its entirety. In order to facilitate sharing of data, we developed SSECRETT as a client-server system so clients can access data remotely through a socket connection. Our prototype system is quite flexible and supports distributed connectome databases and peer to peer data sharing. The connectome volumes can be located on the computer that gives the most convenient access and best network performance. Links can be shared and viewed without ever needing to know where the data actually resides.

3.1 Preprocessing/Dicing

One of the main challenges in handling large datasets is to format the input volume so that any part or resolution is quickly and easily accessible. The volumes are initially stored as a series of large TIFF images. This format is disadvantageous because it is hard to extract an arbitrary sub region from a TIFF file, and neighboring pixels along the z axis are in different files. Also, producing a high-level zoomed out view requires loading almost all the data.

Therefore, in a preprocessing stage we call *dicing*, the input volume is transformed into small three-dimensional blocks. The blocks are recursively sub-sampled and combined to create a hierarchy of resolutions. The structure guarantees that only a small fraction of blocks need to be loaded for any view, high magnification or low. Another advantage of having low resolution blocks available is for progressive updates; low resolution blocks have very fast access and are loaded first. If there is any wait to load high-resolution blocks, SSECRETT will first render a low-resolution image to keep the application responsive.

Our current implementation of dicing is still very time-consuming and I/O bound. For example, it takes about four hours to dice an eight gigabyte raw file. Most of this time is expended reading the images, compressing blocks, and writing files. This time can be reduced, and the algorithm can run in parallel with a distributed file system, but dicing a large volume will still require a significant amount of time. In order to interactively view new stacks, we extended the prototype to simultaneously view multiple volumes at once. A subset of the volume (a stack of images from just scanned slices) can be relatively small, can be diced quickly, and once diced can be visualized as a part of the larger, already processed, volume. This feature has the added advantage that a new stack can be positioned and warped independently, so alignment and stitching can be integrated with the SSECRETT system.

3.2 SSECRETT Server

The SSECRETT server was written in C++ and makes extensive use of the Visualization Toolkit (vtk). Its vocabulary, requests from the client that it recognizes as well as its responses to the client, currently consists of less than 40 messages. Examples of these include messages such as LOAD_FILE (ask the server to load the indicated file), REQUEST (a specific image request), and IMAGE (response from the server to the client indicating the requested image follows). The server can run on Unix/Linux, Mac, or Windows systems.

The current SSECRETT system is designed to support many simultaneous users. It does this by starting a dedicated server for each user. This is the most stable approach but is not the most memory efficient. If multiple users are viewing the same volume, each server will load its own copy of the data. Fortunately, the architecture does not preclude more memory efficient options. A single server can very easily be extended to connect to multiple clients. Another option that takes advantage of modern multi-core processors is to have multiple servers share a single cache that uses system shared memory. Either of these approaches would be useful for collaboration between multiple remote users viewing and discussing the same volume.

The server can be run locally (automatically started up when the client is started up), or remotely, as discussed above. At Harvard, we typically have a server launcher running on several different platforms (with access to different diced datasets), waiting for a client to connect and request a server on that machine. The server launcher acknowledges the client's request (confirming the request has been received), starts a server process, confirms the server is running, and finally passes the connection info to the client. The client can then connect directly to the server to request a dataset, and the visualization and/or analysis of the dataset commences. Meanwhile, the server launcher returns to its listening state, waiting for another client to connect.

The SSECRETT client has the ability to display arbitrary off-axis resampled images. In order to minimize the amount of data sent via the socket, the server resamples the volume and sends just compressed two-dimensional images to the client. This approach avoids sending larger three-dimensional blocks to the client. Additionally, the server does not read or send more information than the client will display given a request. If the client requests an 800×600 image of the entire top slice of a volume, which might be 20k×20k, the server will only read low resolution blocks from disk, only delivering higher resolution when the client can make use of the information.

The prototype has the option to compress the images with JPEG to make them smaller for transmission. The computational cost of compressing images is significant, so compression is skipped when the client is local to the server and network bandwidth is not an issue. With the client in Albany, NY, and the server running in Boston, MA, we can get about 5 frames per second with an 800×600 viewing window (requiring a bandwidth of only 165 KB/s). The frame rate varies considerably as the viewing window size changes; as the window size increases, the frame rate decreases. By comparison, viewing

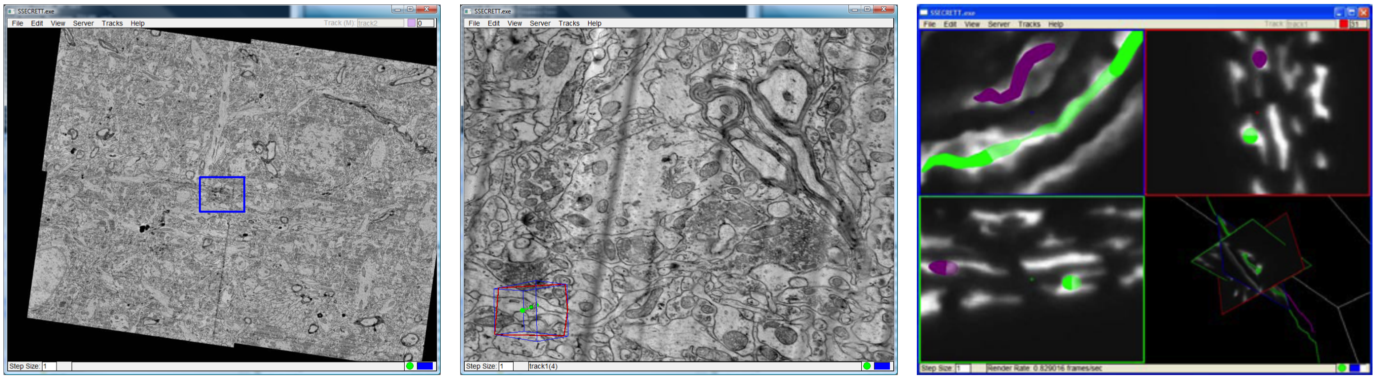

SSECRETT.exe
File Edit View Server Tracks Help
Step Size: 1
track1(4)
Render Rate: 0.829016 frames/sec

Fig. 1: Three different viewing options in the SSECRETT client. Left: A 2D slice view. Middle: A 2D slice view with a 3D view of the slice within the volume. Right: Three orthogonal slice views with a 3D view.

of off-axis images results in negligible degradation in frame rate.

3.3 SSECRETT Client

We have created a simple C++ prototype client viewer, built using FLTK (Fast Light Toolkit; a cross-platform C++ GUI toolkit) and vtk. Our vision is that a variety of clients will connect to the server to request blocks of the volume. SSECRETT manages communication between the client and server via a vocabulary that allows any independent segmentation application to easily access the central server. We regularly run the client on both Mac and Windows systems.

Our prototype client provides a simple manual tracing capability whereby the user drops “breadcrumbs” in the volume via the mouse or keyboard. These “breadcrumbs” are connected to give a track or skeleton tree representing axons or other structures. Tracks can have multiple branches, can be merged with other tracks, and can also include notes associated with the track in general or at specific locations along a track. Furthermore, the notes are easily searchable, so that one user might label regions in the dataset as “weird” or “check this out”, and save the information to a database managing information about the dataset (or just email the tracks file, a small compact format describing just the tracks and notes, to another user). Another user could then pull up the work from the previous researcher, search for the word “weird”, and quickly iterate through the areas in question. Furthermore, a note acts much like a bookmark, saving the camera orientation when the note is created so that returning to a note gives the user an option of viewing it from the same perspective.

The client provides several view options, including a 2D slice view (Figure 1, left), a 2D slice view that includes a small 3D view of the slice within the entire volume (Figure 1, middle), and also a view that gives 3 orthogonal slice views (in addition to a 3D view indicating location of slices within the volume) very similar to the format seen in MRI or CT viewing software (Figure 1, right). In the near future we plan to implement the client using Qt instead of FLTK, which will give the client a more polished feel, as well as allowing greater flexibility and functionality in the interface. This will also enable direct integration of the SSECRETT clinet into the NeuroTrace GUI. Like FLTK, Qt is a cross-platform C++ GUI toolkit.

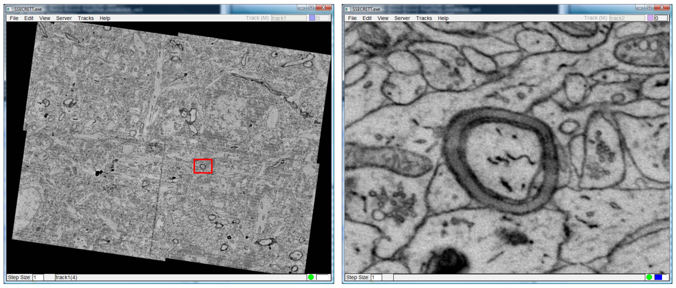


Fig. 2: Zoom capability of the SSECRETT system. Left: A full view of a slice. Right: Zoom in view of the region marked as a red rectangle on the left image.

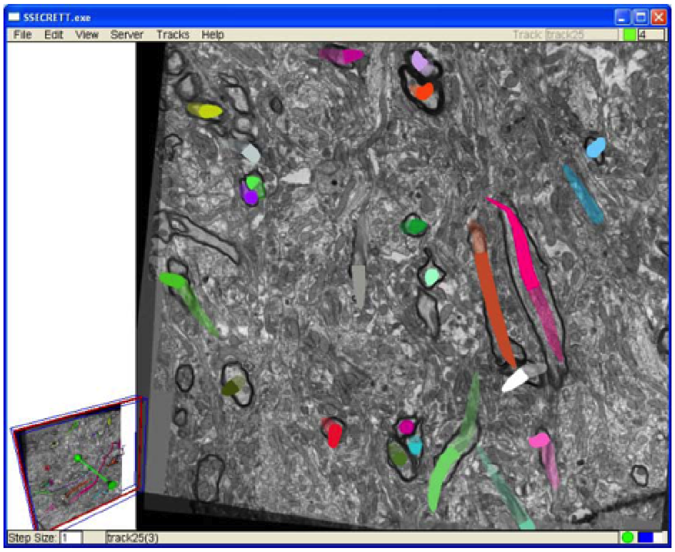


Fig. 3: Manually traced axons using the SSECRETT system.

3.4 Results and Discussion

We have tested the SSECRETT prototype system on several confocal and electron microscope images. Figure 2 shows a full view of an electron microscope slice as well as a zoomed in view, whose location is indicated by the red outline on the full view. The SSECRETT system can progressively transmit coarse-to-fine resolution images per request, and on our local network zooming in and out through stacks of images runs at interactive rates. We have also traced multiple myelinated axons in electron microscope images using the manual tracing capability provided by SSECRETT as shown in Figure 3. Each color segment represents the cross-section of the tube created based on the breadcrumbs dropped by the user, denoting the center axis of the segmented axon. Figure 1 right shows a similar axon tracing result on a stack of confocal microscope images.

In order to test the scalability of the SSECRETT prototype, we simulated a petabyte volume by duplicating an 8 gigabyte volume 50 times along each axis. Although the files were not actually duplicated on disc, the prototype read them as if they were separate files. As expected, the system performed as well on this test as it did on smaller volumes. It is possible that the operating system cached files making the read process more efficient, but using off axis sampling reduced the significance of this effect.

The prototype uncovered two areas where performance of the system can be improved. The first is the time it takes to load and generate the highest-resolution image for a new slice. When no blocks are in cache, it can take up to ten seconds to load all the blocks necessary to generate the full resolution slice. We are considering several ways to deal with this issue. First, we can use smaller leaf blocks. When blocks are small, less data needs to be loaded to generate an image. Second, we could implement a more sophisticated caching algorithm that pre-fetches blocks and anticipates future requests. Third, a distributed database could be run in parallel to take advantage of parallel disk input/output.

The second performance issue we found was that the rendering became slower than desired for very large viewing windows. Viewing of full slices (as shown in Figure 2) with an 800×600 window can be done at a little over 5 frames per second, but that drops to closer to 2 frames per second when using a 1000×1000 window (and is 8 frames per second for a 500×500 window). Performance can be improved by increasing the JPEG compression ratio, but this introduces obvious artifacts. An alternative solution is to use a video compression algorithm like MPEG4 to get better compression because of correlation between successive frames.

The biggest challenge encountered so far is efficiently accessing the original volume to reformat into the multi-resolution diced files. The original data was arranged in a series of 20k×20k TIFF images. We processed slabs of 50 images at a time. This approach is reasonably efficient but breaks down when images are too large. When the size of 50 images overwhelms the memory available on the dicing computer, we switch to dicing 50×50 rows of pixels. This approach, while addressing memory limitations, requires opening and closing each image file many times, and thus still takes too long to process large volumes. The delay is especially pronounced when the raw input files are on a remote file system. The combination of opening files multiple times and the fact that the cubes overlap each other one pixel, causes dicing time to be much longer than a single read of the data. Processing multiple subvolumes helps, but we are working toward an image based dicing strategy that reads and processes images one image at a time.

4 NEUROTRACE

The main goal of NeuroTrace is to provide high-quality 3D volume visualization with user-controllable semi-automatic segmentation capability. Our approach maintains the scalability of the system to large-scale datasets and high-performance parallel computing architectures. An earlier version of NeuroTrace was described in [3]. In this paper, we also discuss the recently added out-of-core volume rendering and batch processing capabilities of the system.

4.1 NeuroTrace Framework

We have implemented NeuroTrace in C++, OpenGL, NVIDIA CUDA and Qt. We use CMake as a cross-platform make tool to be able to easily port the existing application to different operating systems. At the moment, we have built NeuroTrace for Windows and Linux.

The NeuroTrace framework consists of four main conceptual modules:

- *NeuroTrace Core*: This module is the core of the NeuroTrace framework and responsible for data handling and storage, including out-of-core data loading, octree generation and cache management (see Section 4.5).
- *NeuroTrace Segmentation Module*: This module implements the segmentation functionality of NeuroTrace, in our case the active ribbon segmentation algorithm (see Section 4.2).
- *NeuroTrace Visualization Module*: This module is responsible for 3D visualization, including direct volume rendering of EM data, on-demand de-noising and edge-detection, and implicit surface ray-casting of segmented processes (see Section 4.4).
- *NeuroTrace GUI*: This module provides the user interface to the entire application which is completely decoupled from the actual core, segmentation and visualization modules.

Data management is handled in the core. We use a separate loader thread for opening and loading datasets. Whenever the thread has finished loading a new block of data it is interactively added to the visualization. This streaming approach allows us to speed up the initial startup time of the volume renderer by loading and displaying low-resolution blocks first, while the loader thread in the background still reads in the higher resolution blocks. The encapsulation of all data access functions into our data management layer will allow us to switch from local data storage to a network-based file system without any changes in the visualization and segmentation modules.

The *segmentation module* accesses the original data via the NeuroTrace core module. The segmentation results get propagated back and passed on to the interactive visualization system. The segmentation module is treated as a black box by the NeuroTrace framework, meaning that the actual implementation of the segmentation algorithms is not known outside of the module.

The *visualization module* also accesses the data via the NeuroTrace core module. It offers 2D slice and 3D rendering at interactive frame rates, transfer functions, clipping planes, etc. In the case of binary segmentation information, the visualization module also offers two-level volume rendering for segmentation masks.

The *user interface* was developed with the Qt framework. We have implemented a flexible layout using dockable views, which can individually be adjusted by the user, depending on his/her primary requirements. We have implemented a transfer function editor for 1D as well as for 2D transfer functions. Additionally, we offer several adjustable 2D slice as well as 3D views which can be individually activated or hidden, depending on user preferences. We have intentionally kept the user interface decoupled from the rest of the application to be able to switch from one GUI framework to another, if necessary in the future.

Additionally, we keep two tables of all visualization parameters that can be changed in the GUI. One table at the user-interface side, and one table at the renderer-side. This allows us to synchronize the two tables at an arbitrary time in a thread-safe way. Using a separate thread for the user interface keeps the application responsive, even during time-consuming processing tasks. In the future we want to extend NeuroTrace to a client/server application, further extending our scheme of decoupled user-interaction and processing.

4.2 Neural Processes Segmentation

The segmentation method in NeuroTrace combines 2D neuron cross-section segmentation and 3D center point tracking to reconstruct a 3D tubular surface of neural processes. The major benefit of this approach is that the method uses only a small fixed-size 2D subregion of the 3D input data so that it scales well to arbitrarily large datasets. We can treat the subregion retrieval process as a black box using a high-level API (Application Programming Interface), integrated into the NeuroTrace core, without knowing the actual implementation details. For example, we can retrieve a 2D region through the network if the data is located on a server.

For 2D neuron cross-section segmentation, we use the modified active ribbon model originally proposed by Vazquez et al. [12]. The model consists of two deformable interfaces (i.e., level sets) that push or pull against each other based on various internal and external forces until they converge to the inner and outer wall of the cell membrane respectively (Figure 4, ϕ_1 and ϕ_2), which makes the method more robust in noisy and feature-rich electron microscope images than other level set approaches. Some of the forces we use are: the data dependent force to move the interfaces toward the membrane boundary; the ribbon consistency force to keep constant distance between two level set interfaces; the curvature force to maintain the smoothness of the interfaces; and the image correspondence force to robustly initialize the location of the interfaces on subsequent slices. Figure 5 shows several steps of deformation of the active ribbon.

To segment arbitrarily oriented neural processes in 3D space, we have implemented a 3D tracking algorithm that follows the center line

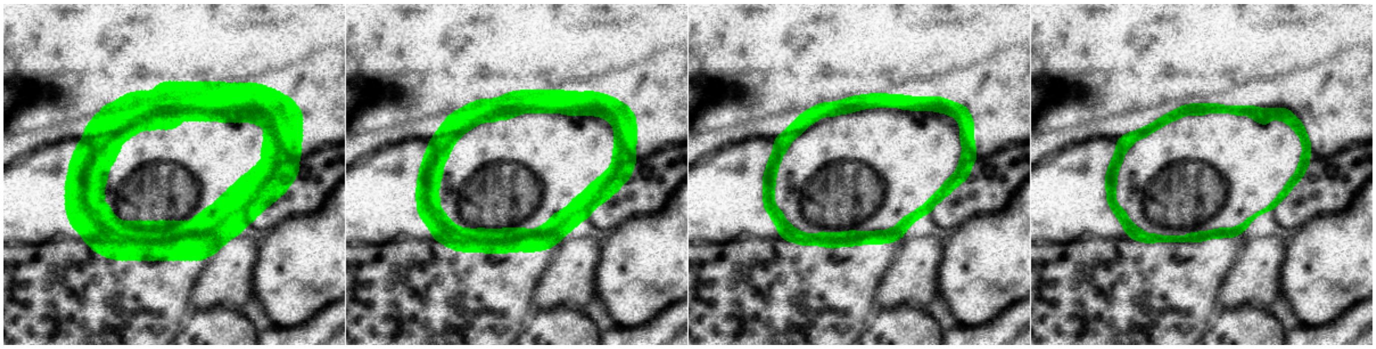


Fig. 5: An example of neural cell membrane segmentation using the active ribbon. From left to right: A user-drawn initial active ribbon (green) is shrinking to the correct membrane boundary.

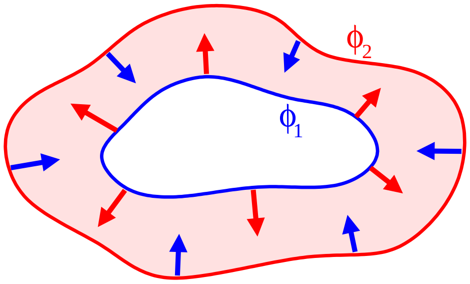


Fig. 4: Active ribbon model consisting of two deformable interfaces (ϕ_1 and ϕ_2) interacting each other.

of the neural process. The main assumptions behind our 3D tracking method are that neural processes are topologically equivalent to 3D tubes, and that the center of each tube cross section lies on the center line of the 3D tube. Therefore, once we find the membrane of the cell cross-section using active ribbon segmentation, we can trace the center of the neural process in 3D space by connecting the center point of each cell cross-section. We propose a two-step method that consists of estimation and correction steps. In the estimation step, the tangent direction V_t at the last center point is computed using a one-sided finite difference method. We also keep the previous tracking direction V_p. The new tracking direction is then the weighted average between those two vectors: $V = \omega V_p + (1-\omega)V_t$ (Figure 6 left). The weight ω controls the amount of history used to determine the current tracking direction to provide smoother transitions between slices. Once we have a new tracking direction, a temporary new center position of the next slice can be estimated by simple extrapolation, and the local frame of the previous slice is then projected onto the new plane defined by the temporary new center and the tracking direction. A new 2D region-of-interest is sampled from the input data for the segmentation process. Finally, in the correction step, the temporary center is replaced by the correct center of the segmented neural membrane. Figure 6 right shows a real example of 3D centerline tracking and axon segmentation using our 3D centerline tracking method.

To speed up the time-consuming active ribbon segmentation process, we have implemented the GPU multiphase level set solver using NVIDIA CUDA. The GPU level set solver employs an iterative, adaptive level set PDE solver using a block-based update scheme. The implementation minimizes the communication between the CPU and GPU and reuses the data on the GPU by safely updating each active block multiple times without updating the active block list. In order to maintain the correct ribbon topology, we need to recompute the signed distance field for each level set when the active block list is updated. To compute image correspondence force, we use a GPU-accelerated deformable image registration method to find a 2D vector field that maps one image to another. More details of this method can be found in [3].

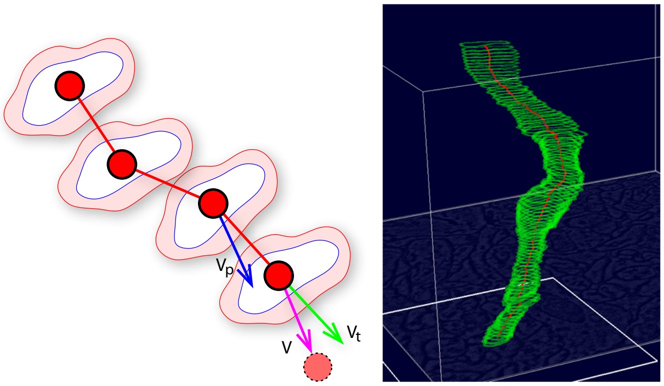


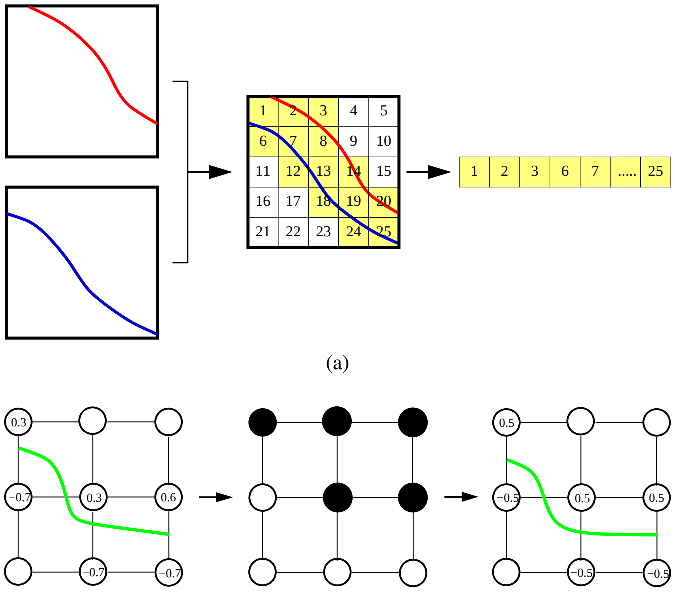
Fig. 6: 3D center line tracking. Left: Weighted extrapolation for neuron cross section center point tracking. Right: Traced axon using NeuroTrace. Green rings are the segmented active ribbons in 2D slices, the red line is the center line of the traced axon.

4.3 Batch and Parallel Processing for Multiple Neural Process Segmentation

Because there are a huge number of neural processes in the dataset and our segmentation method can run automatically without user intervention, we have implemented batch processing for the segmentation of multiple neural processes. The idea is that the user can draw multiple initial level sets, one per neural process, and then let them run sequentially. Once the batch processing ends, the user can examine the result and make any necessary edits to re-run the segmentation to improve the result.

To implement batch processing, we treat each neural process segmentation as an independent process that keeps track of the current status variables and dynamically allocates memory to store the segmentation result. Each segmentation process is then stored on disk as a cache and only the current segmentation process is loaded into main memory. Therefore, a large number of segmentation processes can be launched concurrently as long as each segmentation result can fit into main memory. Batch processing is implemented by switching between segmentation processes and performing the segmentations sequentially.

Another benefit of storing each individual segmentation process is the ability to save the entire workspace and to restore it back later. Due to the enormous data size and numerous neural connections, a complete segmentation of neural processes in a reasonably large input dataset may require days or even weeks of work. To be able to continue the segmentation work for a longer period of time, NeuroTrace provides a way to save the snapshot of the current segmentation on disk. This is done by saving all cached segmentation processes and some additional information, such as a pointer to the input data, the labeled volume, and the current segmentation process index.



(a)

(b)

Fig. 7: Compact representation of the active ribbon. (a) Two floating point images (left) are converted into a binary image (middle), and only the pixels within two level sets, i.e., on the active ribbon, are stored as a list of indices (right). (b) A zero level set defined by a signed distance image (left) is converted into a binary image (middle) and then it is restored back to a floating point distance image, which results in a slight shift from the original zero level set location (right).

To minimize the disk and memory usage, we convert two floating point images of the level set results into a simple but compact representation as shown in Figure 7a. The main idea is that we create a binary image that each pixel value represents either inside or outside of the active ribbon. Then we store only the list of indices for the inside pixels so that we can restore a full binary image later by marking the corresponding pixels as inside. From the recovered binary image, we can restore floating point level set images by assuming that the level set passes through the center of the zero-crossing edge (Figure 7b). This approach may cause a slight shift from the original zero level set, but it does not make a big difference in a realistic setup because we are only interested in pixel-level segmentation results.

To run multiple segmentation processes in parallel, we can distribute processes across different GPUs. The GPUs take care of the assigned processes in parallel. Each GPU accesses the main data server to retrieve the 2D subregion image to run the level set segmentation, but it does not need to store the entire input dataset. Therefore, our parallel segmentation process can easily scale to various hardware configurations and data sizes, such as multi-GPUs in a shared memory system for small and medium size datasets or a GPU cluster system using a fast network for extremely large datasets. The implementation of such a client-server heterogeneous cluster system is subject for future work.

4.4 Volume Visualization

The volume visualization system of NeuroTrace is based on CUDA, NVIDIA's C-like programming environment for GPU programming. We have implemented single-pass raycasting [5] where we use OpenGL for rasterization and CUDA kernels for the actual raycasting. CUDA is a very versatile platform for scientific computing because it enables full utilization of the huge parallel programming power of today's graphics hardware. This allows us to render the volumetric dataset, display the ongoing segmentation, and perform on-the-fly filtering on the visible parts of the volume, while still achieving interactive frame rates. For large data visualization we have integrated support for octree rendering into our raycaster. The required memory management layers are described in Section 4.5.

In NeuroTrace we visualize high-resolution EM data which is extremely dense, heavily textured, and has a low signal-to-noise ratio. To be able to visualize the complex structure of interconnected nerve cells we have added support for on-demand filtering and edge detection of visible parts of the volume. Additionally, we are able to display the results from the neural process segmentation by combining direct volume rendering with implicit surface ray-casting in a single rendering pass. To reduce the required memory footprint, we store the segmentation result in a very compact way.

4.4.1 On-demand Filtering and Edge Detection

Our on-demand filtering and edge detection approach was designed to be able to quickly explore the volume and highlight regions of interest (e.g., myelinated axons). Figures 9a, b, and c show some results in an example EM dataset. For noise removal, we have implemented Gaussian, mean, non-linear median, bilateral and anisotropic diffusion filters in 2D and 3D with variable neighborhood sizes. For EM data, bilateral and diffusion filtering were found to be the most useful. To modulate the opacity of the EM data during volume rendering we us a local histogram-based edge metric [3]. We calculate the histogram difference of two histograms in a voxel's neighborhood several times along different directions to find edges [9]. A high difference between histograms is a good indicator for a large change in brightness, i.e., an edge.

To be able to cope with large datasets our algorithm is based on a hierarchical blocked volume representation (an octree), and a dynamic caching system in GPU and CPU memory. We start by creating a list of all visible blocks for the current viewpoint. In the next step a CUDA kernel performs the filtering on all visible blocks and saves the results in a cache directly in GPU memory. During a final raycasting step the filter/edge values are combined with the original data values to highlight important regions and enhance boundaries in the volume. Less important tissue and more homogeneous regions, on the other hand, are suppressed. To store the on-the-fly computed blocks from the noise-removal or edge-detection pass we allocate two caches directly on the GPU. After all visible blocks for the current viewpoint have been determined, our cache management system automatically decides which blocks actually need to be computed. Blocks that are already present in the cache do not need to be recomputed. Blocks that have already been computed but are not visible in the current viewpoint are kept in the cache for later reuse until the cache reaches its maximum capacity. If cache memory gets low, unused blocks are flushed from the cache in LRU order.

4.4.2 Visualization of Segmented Neural Processes

We concurrently visualize the original EM volume data together with the segmented neural processes in a combined volume rendering. The segmented axons or dendrites are displayed as semi-transparent iso-surfaces, which are rendered in the same raycasting pass as the direct volume rendering of the original data. Keeping our large dataset sizes in mind, we store the segmentation results in a very compact format to circumvent memory restrictions, where each axon is represented as a simple list of elliptical cross-sections. During standard front-to-back raycasting, we compute implicit surfaces from the set of ellipses on-the-fly. At each sample along the ray we test for potential intersections with iso-surfaces. If an intersection is detected, the color and opacity of the iso-surface is composited with the previously accumulated volume rendered part, and volume rendering is continued behind the iso-surface. If the segmentation of the neural processes is performed interactively with user assistance, the axons are continuously updated in the 3D view. This enables the user to stop and adjust the segmentation process whenever necessary. Figure 9d displays several segmented axons in a 3D volume rendering.

4.5 Out-of-core Data Management

The out-of-core data management system of NeuroTrace is comprised of a memory hierarchy that consists of three distinct levels. This hierarchy is illustrated in Figure 8. The first level, the *GPU cache* main-

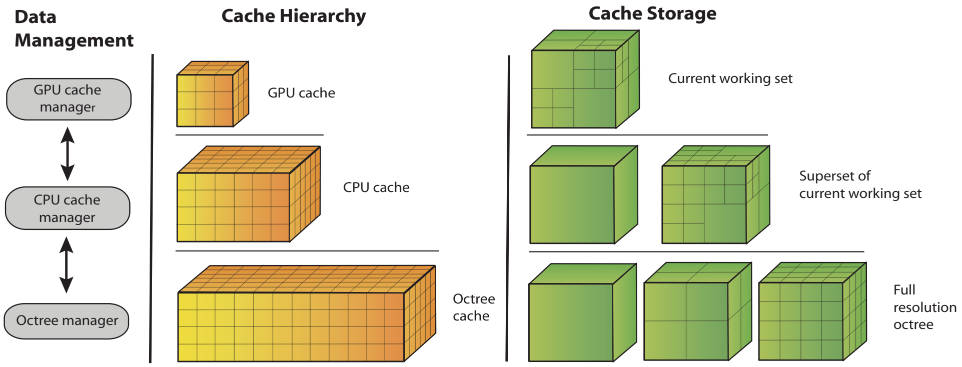


Fig. 8: Three different data management layers are responsible for memory management and caching. The lowest level is the GPU cache, which holds the current working set of blocks. The CPU cache contains a superset of the blocks currently resident on the GPU. The highest level is the octree cache, which stores the entire octree hierarchy in local or network storage.

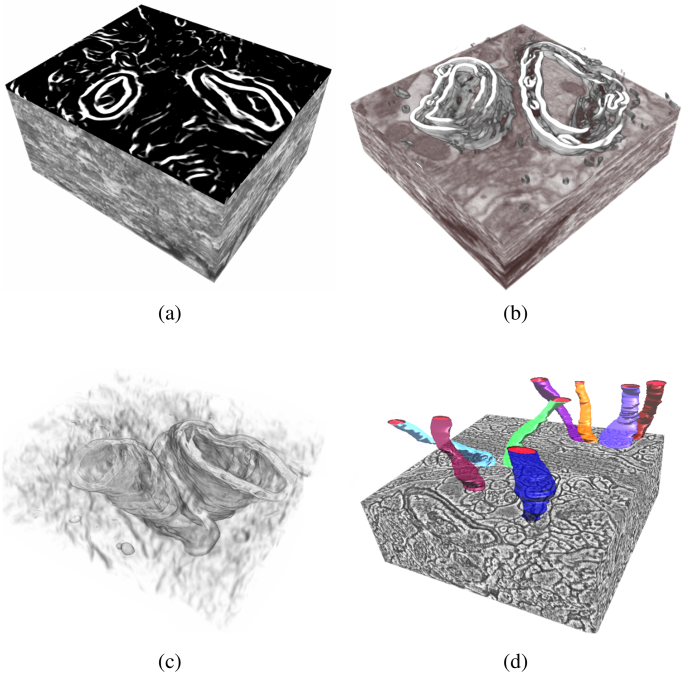


Fig. 9: (a) On-demand edge-detection in an EM dataset. (b) Opacity modulation based on the local-histogram based edge metric. (c) Denoising and edge enhancement. (d) 3D visualization of segmented neural processes.

tained by the *GPU cache manager*, consists of a large 3D cache texture that is always resident in GPU memory and contains all blocks of the volume that are currently needed for raycasting. The cache is managed using a least-recently-used (LRU) scheme, and can freely mix and match blocks of different resolution levels of the volume octree. The second level, the *CPU cache* maintained by the *CPU cache manager*, consists of a large pool of blocks that is a superset of the blocks that are resident in the GPU cache. Blocks in this cache are only flushed when there is not enough space in the cache and otherwise kept as long as possible to enable fast re-downloading of blocks that have been flushed from the GPU cache. This cache is also managed using a LRU scheme. The CPU cache manager can further prefetch blocks from disk that might be needed shortly, so that they are promptly available to the GPU cache manager when requested. The third level, the out-of-core *octree cache* maintained by the *octree manager*, consists of all octree nodes that comprise the entire volume and its multi-resolution hierarchy. All octree nodes of all resolution levels are stored on disk in a single huge binary file. The octree manager is both responsible for initially creating the octree cache when it is not yet available, as well as for loading specific blocks on-demand as they are requested by the CPU cache manager. Loading of requested blocks proceeds in the background using a separate thread, so that the raycaster can proceed while more data are being loaded. While octree nodes are currently loaded from local disk storage, the memory hierarchy we use is conceptually independent of whether blocks are loaded from disk or over the network. We are currently working on extending the octree manager for requesting and loading blocks on-demand over a network connection. All other components of the system will be completely unaware of the load location of the octree nodes.

4.6 Results and Discussion

To evaluate the usability and accuracy of NeuroTrace we have conducted an informal user study, which can be found in [3]. The application was very well received by novice as well as expert users because of its ease of use and accuracy of the automatic segmentation function. Batch processing functionality has added greater flexibility to the existing workflow. Figure 9d shows an example segmentation result.

We achieve interactive frame rates for our on-demand filtering and edge-detection implementation. In the future, however, we would like to extend our on-demand filtering framework. We are currently working on a general multi-resolution histogram framework which would allow us to interactively compare the histograms of different bricks in different resolutions to further enhance our opacity modulated EM data rendering.

A big issue in high-resolution EM data is their high anisotropy, due to the large z-slice distance. This is quite challenging for high-quality visualization of the data. Using multi-resolution approaches, however, we can alleviate this problem by first downsampling along the x and y dimensions of the volume only. After a few downsampling steps, the volume becomes isotropic and we can proceed by downsampling in all three dimensions.

Figure 10 shows a top-down view of a volume that consists of 43 gigabytes of raw data. The entire octree cache for this volume consists of a single file of roughly 75 gigabytes in size, which includes all resolution levels and the duplication of neighbor voxels between adjacent blocks in order to avoid interpolation artifacts at block boundaries. In contrast to only viewing 2D cross sections, viewing the entire volume allows one to perceive 3D structures more easily. In Figure 10, it is also clearly visible that the individual scanned EM slices that constitute the volume are layered on top of each other by the alignment process. Therefore, a full volume is only defined in an interior region of the whole scan, whereas at the borders the originally unaligned slices cover varying regions after alignment. This is an unavoidable side-effect of the acquisition process. The fully defined volume can be cut out easily using axis-aligned clipping planes. However, we usually

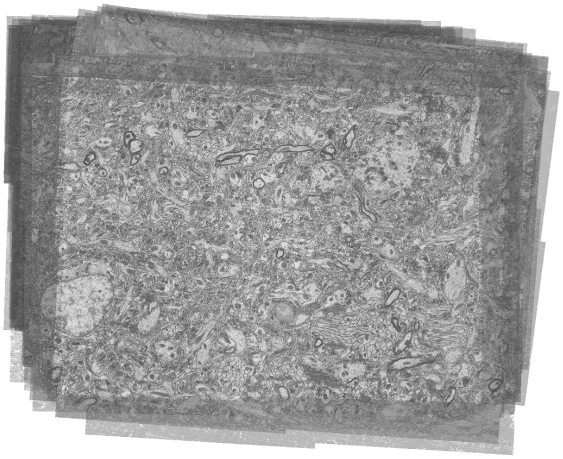


Fig. 10: Interactive overview using semi-transparent volume rendering of a large hippocampus data set comprised of about 43 GB of raw data (resolution $14176 \times 10592 \times 308$). The process that properly aligns the original scanned 2D slices places them in correct relation to each other in 3D, which can clearly be seen at the borders of the volume where varying regions are missing in different slices.

enable the entire unclipped volume, which allows one to view all acquired image data and also allows for visual inspection of alignment accuracy.

Performance for volume rendering using the data set and transfer function depicted in Figure 10 is between 10 and 30 frames per second on a NVIDIA Geforce GTX 285, for a viewport resolution of 800x600. Table 1 gives performance results for different zoomed-in top-down views, which are rendered using the corresponding octree levels that roughly yield a one to one mapping of rendered voxels to displayed pixels. The table also lists what percentage of the displayed octree level and the entire tree, respectively, are actually inside the view frustum. Using a one to one mapping of voxels in the octree to pixels in the output image bounds the amount of memory needed for any given view very effectively. For example, in a top-down view of level 0 of the octree, less than 0.5% of the nodes in this level are actually inside the view frustum. The visible percentage is very low for all zoomed-in views and only increases when the view is zoomed out so much that most or all of the volume becomes visible. The volume is subdivided into blocks of 32^3 voxels. For the $14176 \times 10592 \times 308$ hippocampus volume, the octree is comprised of 10 levels, from a single 32^3 block for the entire volume at the root of the tree (level 9), to $443 \times 331 \times 10$ blocks for the highest-resolution level (level 0).

5 Conclusions

Large-scale image processing and visualization using scalable algorithms and parallel architectures is of immediate interest to neuroscientists to study the connections in the human brain. We have shown that SSECRETT and NeuroTrace introduced powerful solutions using a client-server system and out-of-core data management for efficiently handling large neuroscience datasets. These tools can greatly improve the current data analysis practice that uses manual segmentation and limited visualization functions available in commonly used software tools.

Although we introduce two independent software tools in this article, we envision that both tools will eventually merge into a single program in the near future. Our first step in that direction is to investigate a common data API based on the current out-of-core data management system implemented in NeuroTrace. The common API we are designing will work on different systems and arbitrary data scales seamlessly. We are also working on high-quality 3D volume visualization on a GPU cluster system using a distributed raycasting method. Finally, developing a realtime large-scale neuroscience data acquisition and processing system will be our ambitious long-term research goal.

level	fps	# 32^3 blocks	vis. of level	vis. of tree
0	10	$443 \times 331 \times 10$	- 0.5%	- 0.4%
1	19	$222 \times 166 \times 10$	0.4 - 1.1%	0.1 - 0.2%
2	26	$111 \times 83 \times 10$	2 - 2.3%	0.1%
3	30	$56 \times 42 \times 5$	11 - 55%	0.1 - 0.3%
4	30	$28 \times 21 \times 3$	76 - 100%	0.1%

Table 1: Performance of top-down views of the 43GB hippocampus volume. The view depicted in Figure 10 is a top-down view of octree level 4, which renders at 12 frames per second. Viewport is 800x600.

6 Acknowledgements

This work was supported in part by the National Science Foundation under Grant No. PHY-0835713, the Austrian Research Promotion Agency FFG, Vienna Science and Technology Fund WWTF, the Harvard Initiative in Innovative Computing (IIC), the National Institutes of Health under Grant No. P41-RR12553-10 and U54-EB005149, and through generous support from Microsoft Research and NVIDIA.

References

[1] J. C. Fiala. Reconstruct: a free editor for serial section microscopy. *Journal of Microscopy*, 218(1):52–61, April 2005.

[2] E. Gobbetti, F. Marton, and J. Guitan. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24:797–806, 2008.

[3] 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)*, 2009. to appear.

[4] E. Jurrus, M. Hardy, T. Tasdizen, P. Fletcher, P. Koshevoy, C.-B. Chien, W. Denk, and R. Whitaker. Axon tracking in serial block-face scanning electron microscopy. *Medical Image Analysis (MEDIA)*, 13(1):180–188, February 2009.

[5] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization*, pages 287–292, 2003.

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

[7] J. Lu, J. C. Fiala, and J. W. Lichtman. Semi-automated reconstruction of neural processes from large numbers of fluorescence images. *PLoS ONE*, 4(5):e5655, 05 2009.

[8] J. H. Macke, N. Maack, R. Gupta, W. Denk, B. Schölkopf, and A. Borst. Contour-propagation algorithms for semi-automated reconstruction of neural processes. *Journal of Neuroscience Methods*, 167(2):349–357, 2008.

[9] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):530–549, 2004.

[10] C. Müller, M. Strengert, and T. Ertl. Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV06)*, pages 59–66. Eurographics Association, 2006.

[11] O. Sporns, G. Tononi, and R. Kötter. The human connectome: A structural description of the human brain. *PLoS Computational Biology*, 1(4):e42+, September 2005.

[12] A. Vazquez-Reina, E. Miller, and H. Pfister. Multiphase geometric couplings for the segmentation of neural processes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2020–2027, 2009.