A Scalable Cyberinfrastructure for Interactive Visualization of Terascale Microscopy Data

A. Venkat*, C. Christensen*, A. Gyulassy*, B. Summa^, F. Federer+, A. Angelucci+, V. Pascucci*,

*Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA

+Department of Ophthalmology and Visual Science, Moran Eye Center, University of Utah, Salt Lake City, UT, USA

^Department of Computer Science, Tulane University

Author Email: aniketh.venkat@utah.edu

Abstract— **The goal of the recently emerged field of connectomics is to generate a wiring diagram of the brain at different scales. To identify brain circuitry, neuroscientists use specialized microscopes to perform multichannel imaging of labeled neurons at a very high resolution. CLARITY tissue clearing allows imaging labeled circuits through entire tissue blocks, without the need for tissue sectioning and section-to-section alignment. Imaging the large and complex non-human primate brain with sufficient resolution to identify and disambiguate between axons, in particular, produces massive data, creating great computational challenges to the study of neural circuits. Researchers require novel software capabilities for compiling, stitching, and visualizing large imagery. In this work, we detail the image acquisition process and a hierarchical streaming platform, ViSUS, that enables interactive visualization of these massive multi-volume datasets using a standard desktop computer. The ViSUS visualization framework has previously been shown to be suitable for 3D combustion simulation, climate simulation and visualization of large scale panoramic images. The platform is organized around a hierarchical cache oblivious data layout, called the IDX file format, which enables interactive visualization and exploration in ViSUS, scaling to the largest 3D images. In this paper we showcase the VISUS framework used in an interactive setting with the microscopy data.**

Keywords—3D Imaging; Connectomics; Big data; Terascale Microscopy;

I. Introduction

To date, only a few neuronal circuits have been analyzed comprehensively– the connectivity map of the 302 neurons comprising the entire nervous system of the nematode caenorhabditis elegans being the most notable [6]. While current efforts are being focused on mapping the mouse brain [1] [2], mapping the non-human primate (NHP) brain is rapidly becoming a feasible goal. In particular, the emergence of viral-vector mediated high-resolution fluorescent labeling of neural circuits [3], optical tissue clearing techniques [4], and deep tissue imaging [5] have made it possible to characterize NHP brain wiring at cellular resolution. Understanding human brain function and how neurological and psychiatric diseases affect the brain are just a few fields that would immensely benefit from primate brain maps.

In order to effectively resolve and visualize individual fluorescently labeled axons using 2-photon (2P) or confocal microscopy (axons frequently cross and overlap each other closely through the NHP cortex-see Figure 7(c)), a minimum resolution of 20-40x magnification in the x- and y-planes combined with a z-axis step size of 0.25-1.0 microns between imaging planes is necessary. At these magnifications and z-resolutions one can unambiguously identify continuous neuronal projections and the axonal and dendritic protrusions along them, which identify synaptic points of contact with other cells. While these acquisition parameters allow neuroscientists enough image detail to map brain connectomes, they come at the cost of creating immense amounts of data. In our own 2-photon microscope acquisitions we have found that each 2048x2048 tiff image (8.6MB), which captures a brain area of approximately 0.2 by 0.2 mm^2, provides sufficient image quality for NHP connectome data analysis. Using these parameters to image a tissue block the size of a single macaque monkey cortical area, the primary visual cortex ($\approx$6,000mm^3), would generate approximately 320TB of image data [9], making it nearly impossible to visualize, let alone interact with, the acquired data in its original format.

Traditional 3D visualization tools [10-13] fail to scale when dataset sizes are in the order of several terabytes, resulting in a need for a scalable, high performant software platform that is specifically designed to addresses the big data challenges of neuroscience. In this paper we describe in detail ViSUS, a scalable data analysis and visualization framework for large scale scientific data processing with high performance selective queries. ViSUS Software architecture consists of three major components. First, a lightweight and a fast out-of-core data management framework designed to organize data in an order that exploits the cache hierarchies of many modern data storage architectures. Second, an efficient dataflow framework that allows data to be processed during movement. Third, a portable visualization layer that is designed to scale from mobile devices to powerwall displays with the same code base. These specialized components enable ViSUS to achieve a fast, scalable, and highly portable data processing and visualization environment suitable for exploration and analysis of big data in various fields. In the following sections, we describe the ViSUS software architecture and show how ViSUS is useful in analyzing terascale microscopy images.

II. ViSUS Software Architecture

To provide context, a typical visualization and analysis pipeline is illustrated in Figure 1. We assume that raw data from experiments are available as real-valued, regular samples
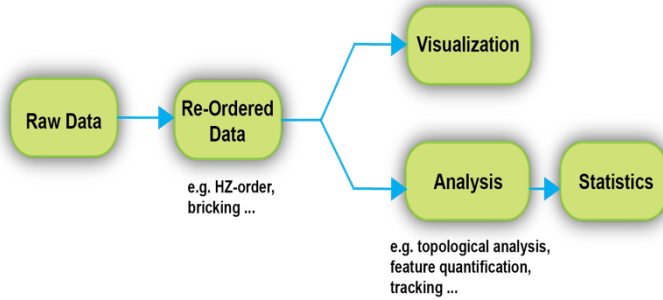
Figure 1. In a traditional out-of-core visualization pipeline where raw data does not fit into memory, the data is often re-ordered to match the capabilities of the rendering engine, for instance, re-ordered into smaller bricks that fit onto GPUs. The hierarchical z-ordered data layout is an integral part of ViSUS platform, and fits into this pipeline. In each case, further analysis, such as feature identification and statistical analysis, often operates on the same data layouts.

of space, possibly varying in time. Due to the large size of datasets, we emphasize that the data samples cannot all be loaded into the main memory. As a result, it is not feasible to use standard implementations of visualization and analysis algorithms on commodity hardware. Figure 2 provides an overview of the software architecture of the ViSUS application framework that we describe next.

Data Access Layer: To achieve data access patterns having minimal performance degradation with external memory storage, it is essential to address the issues of data layout restructuring and algorithm redesign. The ViSUS I/O component (and its generalized database component), in particular focuses on enabling the effective deployment of out-of-core and data streaming algorithms. Algorithmic approaches in this area also yield valuable techniques for parallel and distributed computing. The solution to the out-of-core processing problem is typically divided into two parts: (1) algorithm analysis, to understand data access patterns and, when possible, redesign to maximize data locality; (2) storage of data in secondary memory using a layout consistent with the access patterns of the algorithm, amortizing the cost of individual I/O operations over several memory access operations.

To achieve real-time rates for visualization and/or analysis of extreme scale data, one would commonly seek some form of adaptive level of detail and/or data streaming. By traversing image data hierarchically from the coarse to the fine resolutions and progressively updating output data structures derived from this data, one can provide a framework that allows for real-time access of the data that performs well, even at extreme scale. To maintain efficiency, a storage data layout must satisfy two general requirements: (i) the input hierarchy is traversed from coarse to fine and level by level, so that data in the same level of resolution is accessed at the same time, and (ii) within each resolution level, the regions that are in close geometric proximity are stored as much as possible in close memory locations and also traversed at the same time.

In the past, space filling curves [14] have been used successfully to develop a static indexing scheme that generates a data layout satisfying both of the above requirements for hierarchical traversal (Figure 4). The data access layer of ViSUS employs a hierarchical variant of a Lebesgue space filling curve [15]. The data layout of this curve is commonly referred to as HZ order in the literature. This data access layer has three key features that make it particularly attractive. First, the order of the data is independent of the out-of-core block structure, so that its use in different settings (e.g. local disk access or transmission over a network) does not require any large data reorganization. Second, conversion from the Z-order indexing [16] used in classical database approaches to the ViSUS HZ-order indexing scheme can be implemented with a simple sequence of bit-string manipulations. Third, since there is no data replication, ViSUS avoids the performance penalties associated with guaranteeing consistency especially for dynamic updates. From our test data, we have found that there is only a 27% overhead due to the conversion compared to just copying the raw data which makes this conversion very light. Conversion of large images into the ViSUS format requires no additional storage, compared to the typical 1/3 data increase common for typical tiled image hierarchies. The conversion requires no operations on the pixel data and will outperform even the simplest tiled hierarchies, which require some manipulation of the pixel data.

Beyond the theoretical interest in developing hierarchical indexing schemes for n dimensional space filling curves, ViSUS targets practical applications in out-of-core data analysis and visualization and has been successfully used for direct streaming and real-time remote monitoring of large scale simulations during their executions on IBM BG/L supercomputers at LLNL [17] as well as on Hopper supercomputer at NERSC [18]. The multi-resolution data model used in ViSUS allows adjusting the quality of the visualization depending on the communication speed and on the performance of the local workstation. Owing to the extremely scalable nature of this approach, the same code base is used for a large set of applications while exploiting a wide range of available devices from large power-wall displays to workstations, laptop computers or handheld devices.

Z & HZ ordering: In the two-dimensional case, the Z-order curve can be defined recursively by a Z shape whose vertices are replaced by Z shapes half its size (see Figure 4(a-e)). Given the binary row-major index of a pixel ($i_n...i_1i_0$; $j_n...j_1j_0$) the corresponding Z-order index I is computed by interleaving the indices $I = j_ni_n...j_1i_1j_0i_0$ (see Figure 3 (a) step 1). While Z-order exhibits good locality in all dimensions, it does so only at full resolution and does not support hierarchical access. Instead, ViSUS uses the hierarchical variant, called the HZ-order. This new index changes the standard Z-order to be organized by levels corresponding to a subsampling binary tree, in which each level doubles the number of points in one dimension (see Figure 3 (b)). This pixel order is computed by adding a second step to the index conversion. To compute an HZ-order index $\hat{I}$, the binary representation of a given Z-order index I is shifted to the right until the first 1-bit exits. During the first shift, a 1-bit is added to the left and 0-bits are added in all following shifts (see Figure 3 (a)). This conversion could
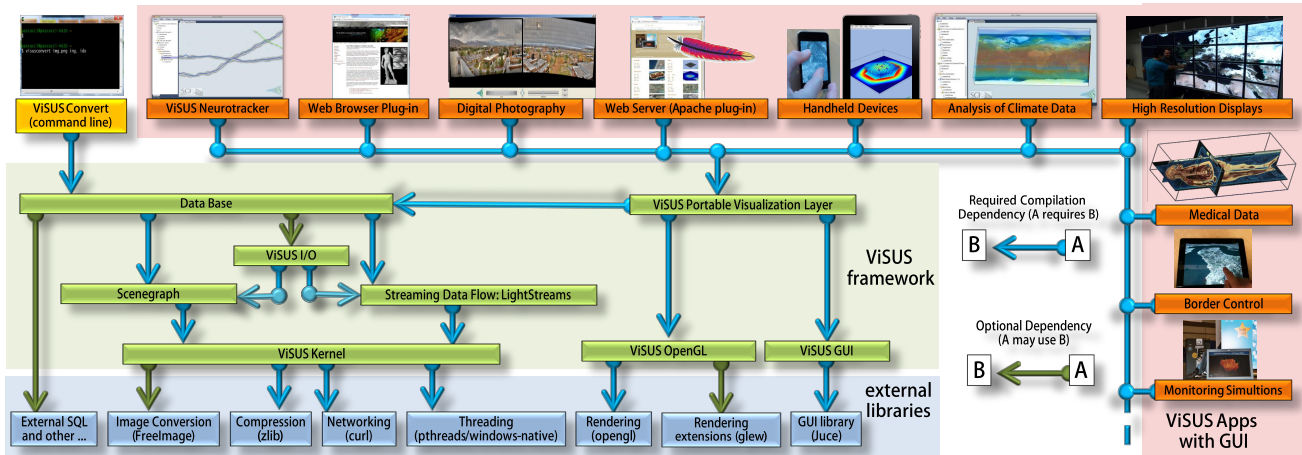
Figure 2. The ViSUS application framework. Arrows denote external and internal dependencies of the main software components. Additionally we show the relationship with several applications that have been successfully developed using this framework.

have a potentially very simple and efficient hardware implementation.
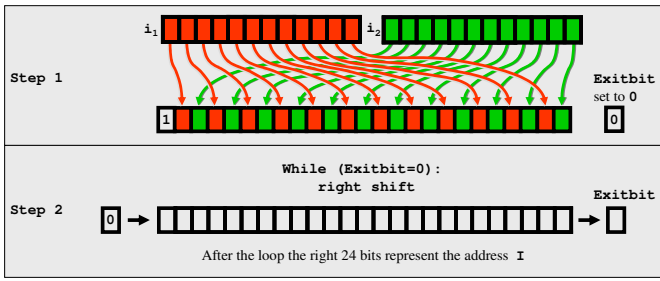
IDX Data format: ViSUS sorts the original data in an HZ-order and groups consecutive samples in blocks of constant size. A sequence of consecutive blocks is grouped into a record and records are clustered in groups, which are organized hierarchically. Each record has a header specifying which of its blocks are actually present and if the data are stored raw or compressed. Groups can miss entire records or subgroups, implying that all their respective blocks and records are missing. The file format is implemented via a header file describing the various parameters (dimension, block size, record size, etc.) and one file per record. The hierarchy of groups is implemented as a hierarchy of directories each containing a predetermined maximum number of subdirectories. The leaves of each directory contain only records. To open a file, one needs only to reconstruct the path of a record and defer its search to the file system. In particular, the path of a record is constructed as follows: we take the HZ-address of the first sample in the record, represent it as a string, and partition it into chunks of characters naming directories, subdirectories, and the record file. Note that, since blocks, records and groups can be missing, one is not restricted to arrays of data that cover the entire index space. In fact, we can easily store even images with different regions sampled at different resolutions.

Efficient Multiresolution Range Queries: One of the key components of ViSUS is the ability to quickly extract rectangular subsets of the input image in a progressive manner. Computing the row-major indices of all samples residing within a given query box is straightforward. However, efficiently calculating their corresponding HZ-indices is not. Transforming each address individually results in a large number of redundant computations by repeatedly converting similar indices. To avoid this overhead, we introduce a recursive access scheme that traverses an image in HZ-order, while concurrently computing the corresponding row-major

indices. This traversal implicitly follows a kd-tree style subdivision, allowing us to quickly skip large portions of the image. To better illustrate the algorithm we will first describe how to recursively traverse an array in plain Z-order. Subsequently, we will discuss how to restrict the traversal to a given query rectangle and finally, how the scheme is adapted to HZ-order. We use a stack containing tuples of type (*split_dimension, I_start, min_i, max_i, min_j, max_j, num_elements*). To start the process, we push the tuple $t0 = (1,0,0,3,0,3,16)$ onto the stack. At each iteration we pop the top-most element t from the stack. If t contains only a single element we output the current *I_start* as HZ-index and fetch the corresponding sample. Otherwise, we split the region represented by t into two pieces along the axis given by split dimension and create the corresponding tuples $t1 = (0,0,0,3,0,1,8)$ and $t2 = (0,8,0,3,2,3,8)$. Note that all elements of $t1$ and $t2$ can be computed from t by simple bit manipulation. In case of a square array, we simply flip the split dimension each time a tuple is split. However, one can also store a specific split order to accommodate rectangular arrays.

To use this algorithm for fast range queries, each tuple is tested against the query box as it comes of the stack and discarded if no overlap exists. Since the row-major indices describing the bounding box of each tuple are computed concurrently, the intersection test is straightforward. Furthermore, the scheme applies, virtually unchanged, to traverse samples in Z-order that sub-sample an array uniformly along each axis, where the sub-sampling rate along each axis could be different. Finally, to adapt the algorithm to HZ-order (see Figure 3 (b)), one exploits the following two important facts:

• One can directly compute the starting HZ-index for each level. For example, in a squared array, level 0 contains one sample and all other levels h contain 2^{h-1} samples. Therefore, the starting HZ-index of level h, I^h_{start}, is 2^{m-h}, where m is the number of bits of the largest HZ-index.

	Binary representation				Level of resolution and its samples in red	
	HZ-order	Z-order	i	j		
0	0 0 0 0	0 0 0 0	0 0	0 0	0	
1	0 0 0 1	1 0 0 0	0 0	1 0	1	
2	0 0 1 0	0 1 0 0	1 0	0 0	2	
3	0 0 1 1	1 1 0 0	1 0	1 0		
4	0 1 0 0	0 0 1 0	0 0	0 1	3	
5	0 1 0 1	0 1 1 0	1 0	0 1		
6	0 1 1 0	1 0 1 0	0 0	1 1		
7	0 1 1 1	1 1 1 0	1 0	1 1		
8	1 0 0 0	0 0 0 1	0 1	0 0	4	
9	1 0 0 1	0 0 1 1	0 1	0 1		
10	1 0 1 0	0 1 0 1	1 1	0 0		
11	1 0 1 1	0 1 1 1	1 1	0 1		
12	1 1 0 0	1 0 0 1	0 1	1 0		
13	1 1 0 1	1 0 1 1	0 1	1 1		
14	1 1 1 0	1 1 0 1	1 1	1 0		
15	1 1 1 1	1 1 1 1	1 1	1 1		

(Sample index labels the left side of the rows.)

Figure 3: (a) Address transformation from row-major index (i; j) to Z-order index I (Step 1) and then to hierarchical Z-order index (Step 2); (b) Levels of the hierarchical Z-order for a 4x4 array. The samples on each level remain ordered by the standard Z-order.

• Within each level, samples are ordered according to plain Z-order and can be traversed with the stack algorithm described above, using the appropriate subsampling rate.

Using these two facts one can iterate through an array in HZ-order by processing one level at a time, adding I^h_{start} to the I_start index of each tuple. In practice, we avoid subdividing the stack tuples to the level of a single sample. Instead, depending on the platform, we choose a parameter n and build a table, with the sequence of Z-order indices for an array with 2^n elements. When running the stack algorithm, each time a tuple t with 2^n elements appears, we loop through the table instead of splitting t. By accessing only the necessary samples in strict HZ-order, the stack-based algorithm guarantees that only the minimal number of disk blocks are touched and each block is loaded exactly once. For progressively refined zooms in a given area, we can apply this algorithm with a minor variation. In particular, one would need to reduce the size of the bounding box represented in a tuple each time it is pushed back into the stack. In this way, even for a progressively refined zoom, one would access only the needed data blocks,
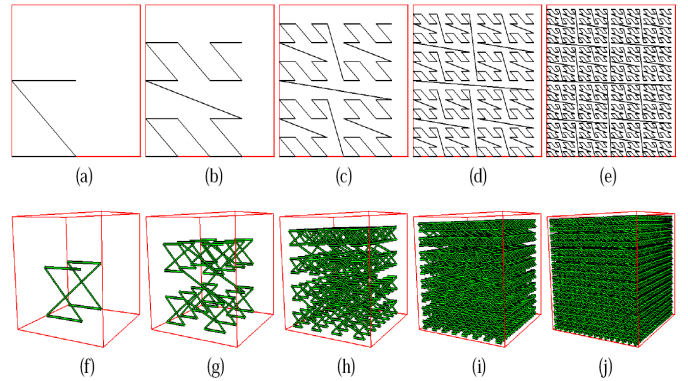

Figure 4: (a-e) The first five levels of resolution of the 2D Lebesgue's space filling curve. (f-j) The first five levels of resolution of the 3D Lebesgue's space filling curve.

each   being   accessed   only   once.

Parallel I/O for Large Scale Simulations: The multi-resolution data layout of ViSUS discussed above is a progressive, linear format and therefore has a write routine that is inherently serial. During the execution of large scale simulations, it would be ideal for each node in the simulation to be able to write its piece of the domain data directly into this layout. Therefore, a parallel write strategy must be employed. Figure 5 illustrates different possible parallel strategies that have been considered. As shown in Figure 5(a), each process can naively write its own data directly to the proper location in a unique underlying binary file. This is inefficient due to the discontinuous access of sparse buffers within memory as well as the large number of small granular, concurrent accesses to the same file. As the data gets large, it becomes disadvantageous to store the entire dataset as a single large file and typically the entire dataset is partitioned into a series of smaller more manageable files. This disjointedness can be used by a parallel write routine. As each simulation process produces simulation data, it can store its piece of the overall dataset locally and pass the data on to an aggregator process. These aggregator processes can be used to gather the individual pieces and composite the entire dataset. In Figure 5(b), each process transmits a contiguous data segment to an intermediate aggregator. Once the aggregator's buffer is filled, the data is written to disk using a single large I/O operation. Figure 5(c), illustrates a strategy where several noncontiguous memory accesses from each process are bundled into a single message. This approach reduces the number of small network messages needed to transfer data to aggregators.

ViSUS Dataflow: In a large scale dataset, even simple manipulations can be very expensive when applied to each variable. Instead, it would be ideal to process the data based on need by pushing data through a processing pipeline as the user interacts with different portions of the data. With the ViSUS multi-resolution layout different regions of the data can be efficiently accessed at varying resolutions. Therefore, different compute modules can be implemented using progressive algorithms to operate on this data stream. Operations such as binning, clustering, or rescaling are trivial to implement on this hierarchy given some known statistics on the data, such as the function value range, etc. These operators can be applied to the
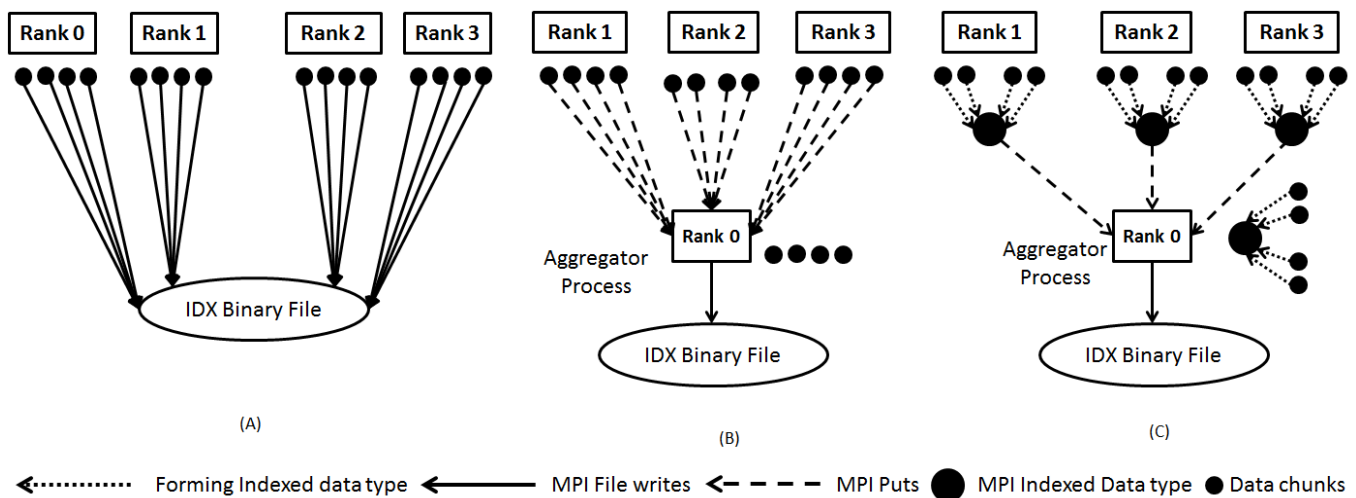
Figure 5: (a) Naive parallel strategy where each process writes its piece of the overall dataset into the underlying file, (b) each process transmits each contiguous data segment to an intermediate aggregator. Once the aggregator's buffer is complete, the data are written to disk, (c) several noncontiguous memory accesses are bundled into a single message to decrease communication overhead.

data stream as-is while the data is moving to the user, progressively refining the operation as more data arrives. More complex operations can also be reformulated to work well using the hierarchy. These adaptive, progressive solutions allow the user to explore a full resolution solution as if it were fully available, without the expense of the full computation. The ViSUS Dataflow facilitates this stream processing model by providing definable modules within a dataflow framework with a well understood API. Figure 6 gives an example of a dataflow for the analysis and visualization of a scientific simulation. This particular example is the dataflow for a microscopy simulation. Each of the dataflow modules provides streaming capability through input and output data ports which can be used in a variety of data transfer/sharing modes.

ViSUS also provides a scene graph hierarchy for both organizing objects in a particular environment, as well as the sharing and inheriting of parameters. Each component in a model is represented by a node in this scene graph and inherits the transformations and environment parameters from its parents. Three-dimensional volume or two-dimensional slice extractors are children of a data set node. As an example of inheritance, a scene graph parameter for a transfer function can be applied to the scene graph node of a data set. If the extractor on this data set does not provide its own transfer function, it will be inherited.

Portable Visualization Layer - ViSUS AppKit: The visualization component of ViSUS was built with the philosophy that a single code base can be designed to run on a variety of platforms and hardware ranging from mobile devices to powerwall displays. To enable this portability, the basic rendering routines were designed to be OpenGL ES compatible. This is a limited subset of OpenGL used primarily for mobile devices. More advanced rendering routines can be enabled if hardware support is available. In this way, the data visualization can scale in quality depending on the available

hardware. Beyond the display of the data, the underlying GUI library can hinder portability to multiple devices. Therefore, ViSUS AppKit provides an abstract GUI interface that currently supports both the Qt and Juce libraries providing lightweight support for mobile platforms such as iOS and Android in addition to major desktop operating systems. ViSUS provides a generic viewer which contains standard visualizations such as slicing, volume rendering and isosurfacing. Additionally, the base system can display 2D and 3D time-varying data. The system considers a 2D dataset as a special case of a slice renderer and therefore the same code base is used to render both 2D and 3D data.

Webserver and Plugin: ViSUS has been extended to support a client-server model in addition to the traditional viewer. The ViSUS server can be used as a standalone application or a web server plugin module. The ViSUS server uses HTTP (a stateless protocol) in order to support many clients. A traditional client/server infrastructure, where the client established and maintained a stable connection to the server, can only handle a limited number of clients robustly. Using HTTP, the ViSUS server can scale to thousands of connections. The ViSUS client keeps a number (normally 48) of connections alive in a pool using the "keep-alive" option of HTTP. The use of lossy or lossless compression is configurable by the user. For example, ViSUS supports JPEG and EXR for lossy compression of byte and floating point data, respectively. The ViSUS server is an open client/server architecture, therefore it is possible to port the plugin to any web server which supports a C++ module (i.e., Apache, IIS). The ViSUS client can be enabled to cache data to local memory or to disk. In this way, a client can minimize transfer time by referencing data already sent, as well as having the ability to work offline if the server becomes unreachable. The ViSUS portable visualization framework (Appkit) also has the ability to be compiled as a Google Chrome, Microsoft Edge, or Mozilla Firefox web browser plugin. This allows a ViSUS framework
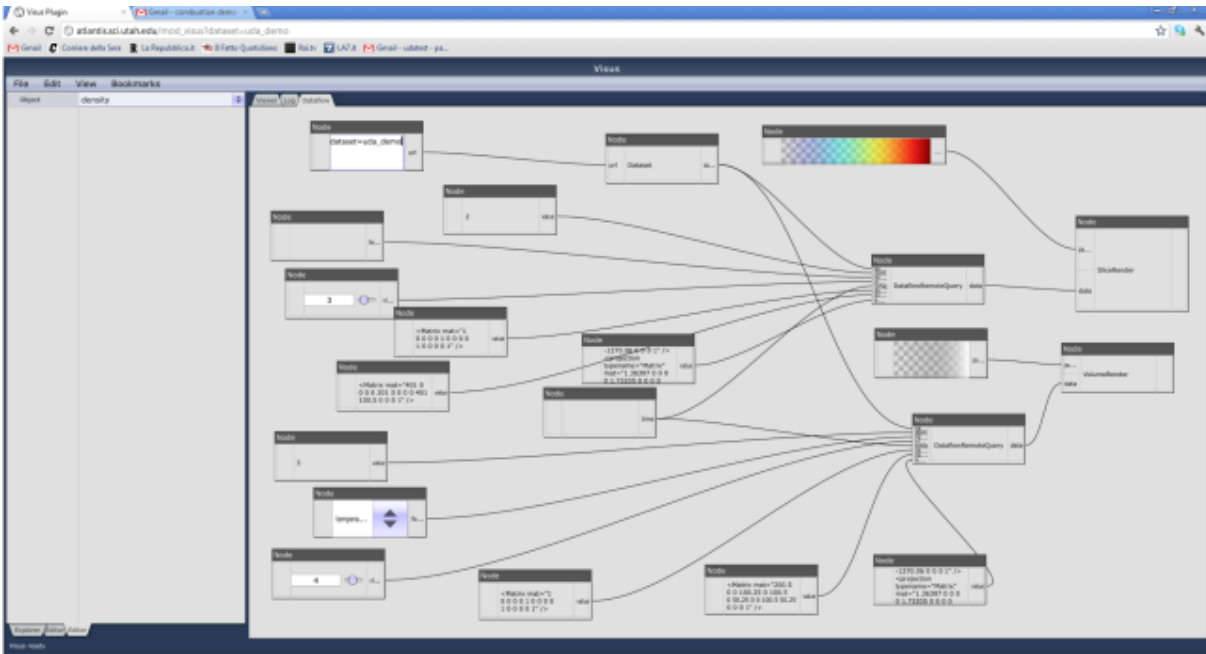
Figure 6. Several dataflow modules chained together to provide a light and a flexible stream processing capability.

based viewer to be easily integrated into web visualization portals.

III. APPLICATION: MICROSCOPY

The ViSUS application framework has been used at the Moran Eye Centre, University of Utah to visualize 2-photon2P microscope acquired images of axons labeled with GFP (through intracortical injections of AAV9-GFP) and blood vessels labeled with Alexa594-conjugated tomato lectin through transparent Clarity-treated blocks (~60mm3) of marmoset monkey V1. These scans yielded a 3TB dataset with sufficient imaging detail for tracing. The acquired images were manually transferred to a local ViSUS server where the data was converted and stored in our multi-resolution IDX format. Storing the data on a server accessible to the ViSUS viewer applications allows the user to rapidly open the image stacks into a single 3D volume, which can be viewed and manually aligned in x, y and z (Figure 7(a)). In tissue volumes containing only sparse GFP-labeled axons (green channel), we used the stained blood vessels (red channel), which are larger than axons and densely weave throughout the cortex, for volume-to-volume alignment (Figure 7(b)). After this, switching to the green channel (which is simultaneously imaged using multi-channel 2P acquisition) yielded perfectly aligned sparse GFP-labeled axons (Figure 7(c)).

IV. FUTURE WORK

While ViSUS infrastructure enables interactive visualization and analysis of terascale microscopy data, the vast amount of data presents new problems in the acquisition process. Based on our current acquisition parameters, we have estimated that scanning an entire mouse brain would result in a 31TB dataset. While big, this dataset is small compared to the estimated 320TB that would result from imaging labeled circuits at mesoscopic scale just in the macaque monkey primary visual cortex (~6,000mm³). Moreover, a slight increase in resolution would quickly double or quadruple the dataset size. A computer with hardware powerful enough to process and visualize that many images is beyond the scope of a typical laboratory budget, requiring frequent human intervention to offload the data to a server and continue the acquisition. Further, once the acquisition and the copying process is completed, it is required to manually convert millions of images into a multiresolution data format before the data can be visualized, causing additional delays. To address this sizeable issue of data transfer and conversion that currently hinders our ability to interactively explore and analyze connectomics data, we plan to expand the ViSUS streaming platform, by introducing a number of web services that convert in real time 2D images generated by the microscope to IDX data format. Our new acquisition system saves significant processing time by amortizing the conversion and copying effort, allowing the microscope to continuously operate to acquire massive datasets. As a consequence, investigators can remotely monitor the progress of their acquisitions which can be invaluable for immediate analysis of critical images. In addition, we plan to incorporate real time diagnostics into ViSUS to detect variations in the acquired image quality which would be particularly useful during a long acquisition to identify and rectify unforeseen issues. We envision our cyberinfrastructure to perform all these operations without adding additional delays providing an end-to-end scalable solution for terascale microscopy.

As data sizes continue to grow, the relative ability of a user to manually trace neurons decreases. Automated analysis will become an integral part in obtaining full-scale wiring diagrams. These trends inevitably require migration of data to HPC resources and deployment of algorithms in an HPC setting. The ViSUS infrastructure has already been shown to perform as a state-of-the-art I/O infrastructure for large scale simulations,

and we plan to add the functionality needed to buffer and migrate data from microscopes to the remote HPC setting, to design and schedule analysis workflows, and achieve long-term storage of the raw imagery.

V. CONCLUSIONS

Interactive visualization and data exploration are an indispensable part of hypothesis testing and knowledge discovery. Like most fields, visualization faces substantial scientific data management challenges that are the result of growth in size and complexity of the data being produced by simulations and collected from experiments. In the context of neuroscience, the large size and complexity of the NHP brain produces massive amounts of imaging data, creating great computational challenges to the study of neural circuits. To address these challenges, neuroscientists require novel software capabilities for compiling, stitching, and visualizing large imagery. With data of massive scale, it is often useful to perform a multiresolution analysis, working first with a smaller, coarser version of the data, then progressively refining the analysis as interesting features are revealed. With ViSUS we showed that a space-filling curve model has proven to be highly efficient for interactive analysis of massive neuroscience data. Topological methods for multi-scale,

quantitative feature detection and analysis have also been demonstrated to be highly effective by providing intermediate concise descriptions of the data. By providing a higher level abstraction, they enable scientists to explore feature definitions interactively even if the raw data is prohibitively large in size.
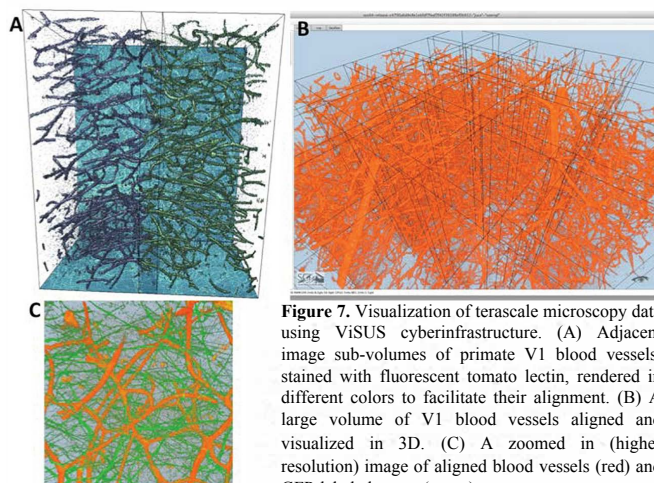

Figure 7. Visualization of terascale microscopy data using ViSUS cyberinfrastructure. (A) Adjacent image sub-volumes of primate V1 blood vessels, stained with fluorescent tomato lectin, rendered in different colors to facilitate their alignment. (B) A large volume of V1 blood vessels aligned and visualized in 3D. (C) A zoomed in (higher resolution) image of aligned blood vessels (red) and GFP-labeled axons (green).

REFERENCES

[1] Briggman, K.L., Helmstaedter, M. & Denk, W. Wiring specificity in the direction-selectivity circuit of the retina. Nature 471, 183–188 (2011).

[2] Bock, D.D. et al. Network anatomy and in vivo physiology of visual cortical neurons. Nature 471, 177–182 (2011).

[3] Luo, L., Callaway, E. M. & Svoboda, K. (2008). Genetic dissection of neural circuits. Neuron 57, 634-660.

[4] Chung, K, Wallace, J, Kim, SY, Kalyanasundaram, S, Andalman, AS, Davidson, TJ, Mirzabekov, JJ, Zalocusky, KA, Mattis, J, Denisin, AK, Pak, S, Bernstein, H, Ramakrishnan, C, Grosenick, L, Gradinaru, V & Deisseroth, K. (2013). Structural and molecular interrogation of intact biological systems. *Nature* 497**,** 332-337.

[5] Denk W.; Strickler J.; Webb W. (1990). "Two-photon laser scanning fluorescence microscopy". *Science* 248 (4951): 73–6.

[6] White, J.G., Southgate, E., Thomson, J.N. & Brenner, S. The structure of the nervous system of the nematode Caenorhabditis elegans. Philos. Trans. R. Soc. Lond. B Biol. Sci. 314, 1–340 (1986).

[7] Economo MN, Clack NG, Lavis LD, Gerfen CR, Svoboda K, Myers EW, Chandrashekar J A platform for brain-wide imaging and reconstruction of individual neurons. Elife. 2016 5:e10566.

[8] Ragan T, Kadiri LR, Venkataraju KU, Bahlmann K, Sutin J, Taranda J, Arganda-Carreras I, Kim Y, Seung HS, Osten P. 2012. Serial two-photon tomography for automated ex vivo mouse brain imaging. Nature Methods 9:255–258.

[9] Christensen, C, Federer, F, Gooch, A, Merlin, S, Pascucci, V & Angelucci, A. (2015). Large scale imaging and 3D visualization of long-range circuits in CLARITY-treated primate visual cortex. . *Soc Neurosci Abstr Online* 598.19.

[10] Clendenon, J.L., Phillips, C.L., Sandoval, R.M., Fang, S. & Dunn, K.W. Voxx: a PC-based, near real-time volume rendering system for biological microscopy. Am. J. Physiol. Cell Physiol. 282, C213–C218 (2002).

[11] Swedlow, J.R., Goldberg, I., Brauner, E. & Sorger, P.K. Informatics and quantitative analysis in biological imaging. Science 300, 100–102 (2003).

[12] Abràmoff, M.D., Magalhães, P.J. & Ram, S.J. Image processing with ImageJ. Biophotonics Internatl. 11, 36–42 (2004).

[13] Sommer, C., Straehle, C., Kothe, U. & Hamprecht, F.A. Ilastik: Interactive learning and segmentation toolkit. Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on 230–233 (2011).

[14] Hans Sagan. Space-Filling Curves. Springer-Verlag, New York, NY, 1994.

[15] Valerio Pascucci and Randall J. Frank. Global static indexing for real-time exploration of very large regular grids. In Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM), pages 2–2, New York, NY, USA, 2001. ACM Press.

[16] J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. Lecture Notes in Computer Science, 1832:20, 2000.

[17] S. Kumar, V. Pascucci, V. Vishwanath, P. Carns, R. Latham, T. Peterka, M. Papka, and R. Ross. Towards parallel access of multi-dimensional, multiresolution scientific data. In Proceedings of 2010 Petascale Data Storage Workshop, November 2010.

[18] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout. Pidx: Efficient parallel i/o for multi-resolution multi-dimensional scientific datasets. InProceedings of IEEE Cluster 2011, September 2011.