ConnectomeExplorer: Query-Guided Visual Analysis of Large Volumetric Neuroscience Data

Johanna Beyer, Ali Al-Awami, Narayanan Kasthuri, Jeff W. Lichtman, Hanspeter Pfister, and Markus Hadwiger

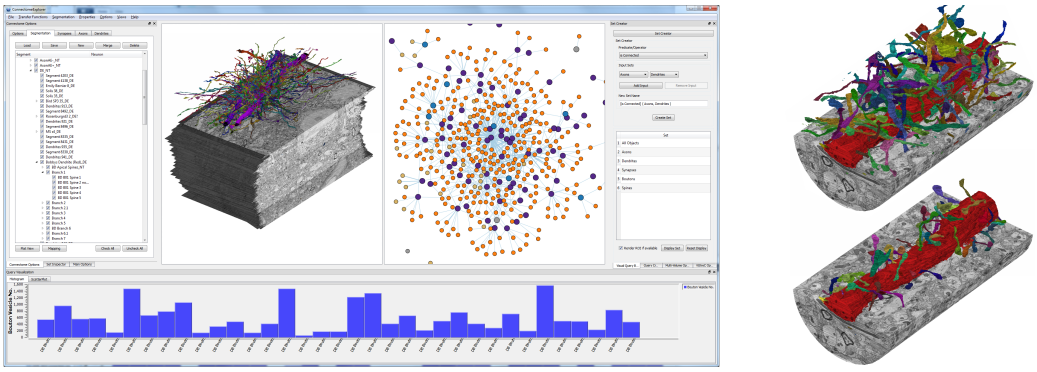


Fig. 1. ***ConnectomeExplorer*** enables the interactive visual analysis of large data volumes in connectomics research, integrating 3D volume data of brain tissue and segmented objects, connectivity of cells and neurites, and additional meta data. A powerful query algebra allows neuroscientists to pose domain-specific questions in an intuitive manner, and to interactively analyze the results. We show a teravoxel volume via (left) a tree widget of segmented objects, a 3D volume view, a connectivity graph of neurites (axons and dendrites), a visual query builder for dynamic query specification, and a statistics view at the bottom; (right) visualization of a dynamically specified region of interest (top: all neuronal objects in a cylindrical region; bottom: only the spines of the red dendrite).

Abstract—This paper presents *ConnectomeExplorer*, an application for the interactive exploration and query-guided visual analysis of large volumetric electron microscopy (EM) data sets in connectomics research. Our system incorporates a knowledge-based query algebra that supports the interactive specification of dynamically evaluated queries, which enable neuroscientists to pose and answer domain-specific questions in an intuitive manner. Queries are built step by step in a visual query builder, building more complex queries from combinations of simpler queries. Our application is based on a scalable volume visualization framework that scales to multiple volumes of several teravoxels each, enabling the concurrent visualization and querying of the original EM volume, additional segmentation volumes, neuronal connectivity, and additional meta data comprising a variety of neuronal data attributes. We evaluate our application on a data set of roughly one terabyte of EM data and 750 GB of segmentation data, containing over 4,000 segmented structures and 1,000 synapses. We demonstrate typical use-case scenarios of our collaborators in neuroscience, where our system has enabled them to answer specific scientific questions using interactive querying and analysis on the full-size data for the first time.

Index Terms—Connectomics, neuroscience, query algebra, visual knowledge discovery, petascale volume analysis

1 Introduction

Reconstructing the anatomical and functional connectivity within the brain has become one of the most active research areas in neuroscience. By ultimately mapping and deciphering a human's entire *connectome* [39], i.e., the full "wiring diagram" of the brain comprising billions of neurons and their interconnections, scientists hope to gain an understanding of how the brain develops and functions, and how pathologies develop or can be treated. To support this goal, high-throughput methods for neural imaging have been developed that enable scientists to acquire imaging data at unprecedented speed and resolution. However, the extreme size of the resulting electron microscopy (EM) imaging volumes and the complexity of the structures contained in these data present significant challenges.

Most previous research has focused on the actual image acquisition and subsequent semi-automatic or fully automatic segmentation of EM slice stacks. However, less research has focused on the next step: on how to enable efficient neuroscientific analysis of volume and segmentation data of this size and complexity. Neuroscientists now have huge collections of high-resolution EM volumes and their segmentations, but no efficient means for analyzing them or directly answering high-level domain-specific questions. Most current tools only provide 2D visualizations of subsets of the EM data or the corresponding segmentations. The latter are either provided at the voxel level or—more commonly—consist of extracted geometry. To the best of our knowledge, no existing tool offers interactive 3D visualization of multiple teravoxel volumes, e.g., EM and segmentation volumes, while at the same time allowing scientists to dynamically explore and analyze the entire data set by posing domain-specific questions in an intuitive way.

In this paper, we introduce *ConnectomeExplorer*, which is an integrated application for the exploration and query-guided visual analysis of large EM volumes in connectomics research. Our system is based on a scalable volume visualization framework that scales to petascale data from high-throughput microscopy data streams [5, 22], which, however, did not support data analysis or querying. We introduce a knowledge-based query algebra that enables analysis via interac-

- *Johanna Beyer is with King Abdullah University of Science and Technology (KAUST). E-mail: johanna.m.beyer@gmail.com.*
- *Ali Al-Awami is with King Abdullah University of Science and Technology (KAUST). E-mail: ali.awami@kaust.edu.sa.*
- *Markus Hadwiger is with King Abdullah University of Science and Technology (KAUST). E-mail: markus.hadwiger@kaust.edu.sa.*
- *Narayanan Kasthuri is with the Center for Brain Science at Harvard University. E-mail: bobby.kasthuri@gmail.com.*
- *Jeff W. Lichtman is with the Center for Brain Science at Harvard University. E-mail: jeff@mcb.harvard.edu.*
- *Hanspeter Pfister is with the School of Engineering and Applied Sciences at Harvard University. E-mail: pfister@seas.harvard.edu.*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

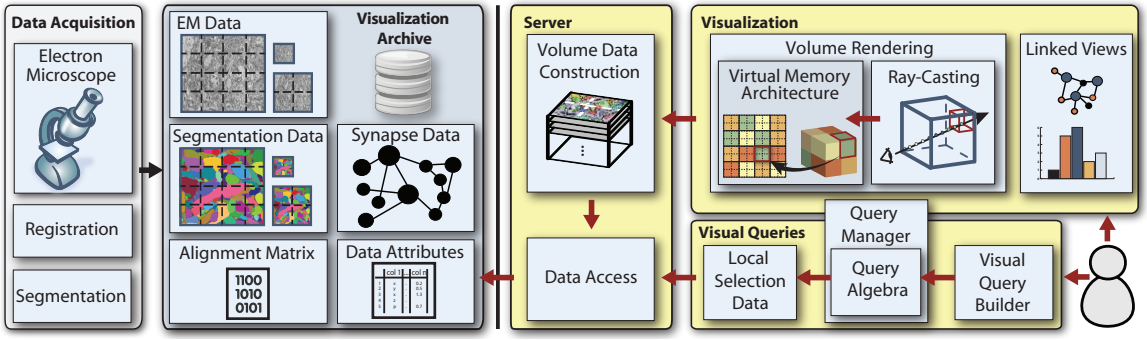


Fig. 2. **System overview.** Our system consists of two main parts: data-driven modules (left, light gray) are triggered by image acquisition, while visualization/user-driven modules (right, yellow) are active at run time. All generated data are stored in the visualization archive. At run time, the user initializes data requests either by specifying a dynamic query or by directly interacting with the visualization. A visual query builder allows users to intuitively specify queries which are translated into a powerful query algebra and evaluated. Results are shown in different linked visualizations.

tively specified dynamic queries that allow neuroscientists to answer domain-specific questions. The main contributions of this paper are:

- An integrated system for the interactive analysis and 3D visualization of large-scale neuroscience data, based on demand-driven processing and interactive, dynamically evaluated user queries.
- A query (set) algebra that provides scientists in connectomics research with an intuitive way of specifying dynamic, high-level queries that can be evaluated on the full data size. Complex queries can be built by hierarchically combining simpler queries.
- An intuitive user interface that allows specifying combinations of queries in different domains (spatial, connectivity/topological, abstract/attribute-based), and to concurrently explore and analyze the results in multiple linked views. Both the queries and the linked views also provide basic statistical analysis.

We illustrate the usefulness of *ConnectomeExplorer* in practice via real use-case scenarios of our collaborators in neuroscience.

2 Related Work

Neuroscience and connectomics. A very good introduction to connectomics and recent developments is given by Seung [39], also highlighting advances in high-throughput, high-resolution electronic imaging. Lichtman and Denk [31] describe the challenges in reaching the ultimate goal of connectomics—understanding the relation between function and structure in the brain. Bock et al. [8] present a powerful example of how EM circuit reconstruction allows determining the relationship between structure and function of the visual cortex.

Segmentation and annotation tools for neuroscience. Segmentation and tracing of neuronal structures ranges from manual [17] or semi-automatic [24, 27, 37] to fully automatic segmentation algorithms [25, 28]. However, only a handful of tools are actually publicly available to the neuroscience community. In Eyewire [1], users trace neurons in the retina in an online game setting. Mojo [29, 37] offers fast proof-reading of segmented EM slices. NeuroTrace [26] supports semi-automatic segmentation of neurites with concurrent 3D visualization. CATMAID [38] and the Viking viewer [4] are collaborative annotation environments for skeleton extraction in terabyte data sets.

Visualization of microscopy data. Volume visualization of microscopy data is an ongoing research topic due to their inherent visual complexity. Different techniques for displaying features in dense EM image stacks have been proposed, including multi-dimensional transfer functions [45], local variance-based transfer functions [35], and view-dependent on-demand filtering and edge enhancement [27].

Neuroscience ontologies. Several groups have worked on neuroanatomy ontologies [36] and how to leverage this knowledge for visualization and data analysis [7, 20, 30]. Gerhard et al. [20] introduce the Connectome Viewer Toolkit, a framework for multi-modal data management as well as visualization and analysis of macroscopic neuronal structures, pathways and brain region connectivity. Kuß et al. [30] propose a system for high-level ontology-based queries on a bee brain atlas that supports a set of pre-defined visualization queries.

Visual analysis in neuroscience. Braingazer [9] is a system for visually analyzing a Drosophila (fruit fly) brain database. It supports 3D visualization of confocal microscopy data together with annotated anatomical structures. It allows users to interactively query the data based on semantic and spatial relationships, but it does not provide a general way for specifying new domain-specific questions. It also does not support the fully dynamic computation of the required constructs. Neuron Navigator [32] provides an interface to a 3D neuron image database to analyze the connectivity of the Drosophila brain. It offers a textual query interface based on binary operators to select and display objects and anatomical structures. De Leeuw et al. [13] introduce the Argos system to analyze and visualize large 3D microscopy image collections in a combination of offline and interactive processes. Sherbondy et al. [40] use dynamic queries based on volumes of interest and pre-computed pathways to explore the connectivity between brain regions. Most systems for visually analyzing neuroscience data sets are based on atlases, i.e., they use databases of pre-defined structures. Many systems also require the pre-computation of some query attributes, such as proximity or distances. In contrast, our framework supports the fully dynamic analysis of non-templated volumetric data sets without requiring costly pre-computations that do not scale to the petavoxel data sizes that we are targeting. We allow specifying any number of new dynamic queries, which are evaluated only on demand. We compare different tools that are currently used in connectomics research with *ConnectomeExplorer* in Table 4 (Sec. 7).

Dynamic and visual queries. Prominent techniques are dynamic queries [3], high-dimensional brushing and linking [34], and interactive visual queries [14]. Shneiderman [41] gives a good introduction to dynamic queries in the context of visual analysis and information seeking. Catarci et al. [11] present a survey on visual query systems for databases. More recently, Gergen et al. [21] describe design lessons for developing visual information-seeking systems, and Heer and Shneiderman [23] introduce a taxonomy of interactive dynamics for visual analysis. A visual interface for exploration and analysis of large multi-dimensional databases is available in Polaris [44], while DEX [43] focuses on query-driven scientific visualization of large data sets. Nitelight [42] introduces a visual approach for semantic query design, and ImMens [33] is a recent visual query system for real-time analysis of very large data sets, supporting brushing & linking and scalable visual summaries. A general introduction to the topics of relational algebra and databases is given by Garcia-Molina et al. [19].

Volume rendering. Many large-scale volume renderers employ octree bricking schemes, and often perform octree traversal on the GPU [12, 16]. The volume rendering system of *ConnectomeExplorer* is based on previous work on GPU ray-casting of petascale EM data via a multi-level, multi-resolution virtual memory architecture [22], which also supports streaming of microscopy image data and visualization-driven construction of volume sub-blocks. This system has been extended to segmentation volumes [5], but did not support analyzing or querying the data. Cai and Sakas [10] present different ways of rendering multi-modal data. Beyer et al. [6] used multi-modal volume rendering in the context of neurosurgical applications.

3 Application Overview

ConnectomeExplorer was developed in collaboration with neuroscientists working in the field of connectomics to support them in analyzing the detailed interconnections of neuronal structures in the mammalian brain at the individual nanometer scale. Each neuron processes and transmits information by forming connections (i.e., synapses) to other neurons. A typical neuron consists of a *cell body (soma)*, an *axon*, and multiple *dendrites* [39]. Roughly speaking, an axon corresponds to the output of the neuron, and its dendrites correspond to all its inputs. Axons are long and narrow tubular structures that conduct electrical impulses away from the neuron's cell body when the neuron *spikes*, while a dendrite is a treelike extension of a neuron that receives electrical impulses from many other neurons. Axons make connections with dendrites at *synapses*, which comprise the synaptic cleft between an *axon terminal* (or *bouton*) and a *post-synaptic density* of the main part of a dendrite or a *dendritic spine*. Neuroscientists are interested in trends and correlations as well as individual neuronal structures and synapses and their detailed attributes. For example, the location of a synapse on a dendrite (e.g., whether or not it is on a spine) may be an indicator for whether the entire cell is *excitatory* or *inhibitory*, i.e., if it increases or decreases the likelihood of connected neurons to spike.

In this section, we give an overview of the basic components of our system, from data acquisition and management to the integration of visual analysis and exploration tools. A high-level overview is depicted in Fig. 2. Table 1 gives a comprehensive list of *object types* and pre-defined *sets* of objects that we support as basic inputs to queries.

3.1 Data Acquisition

Our collaborators cut blocks of brain tissue into ultra-thin slices of 25-50 nm using an advanced microtome, and image them with a scanning electron microscope (SEM) at a resolution of 3-5 nm. Each EM slice comprises multiple microscope images tiles (e.g., $12,000 \times 12,000$ pixels each), which are aligned and stitched into a much larger 3D volume [22]. The left part of Fig. 2 shows the data acquisition modules of our pipeline. For segmentation, our collaborators have used several different approaches, ranging from complete manual tracing of individual structures to automatic tracing [28]. *ConnectomeExplorer* also supports manual editing of segmentation masks at the voxel level. All other features and meta data, such as labels and synapse locations, are currently annotated completely manually and stored in tabular format.

3.2 Data Management

Our data managment approach comprises two main parts: a data-driven part, which is triggered by actual data acquisition and generation, and a visualization-driven part, which is active at run time. We generate tiled 2D mipmaps of the acquired raw image data, which subsequently serve as the source for all data requests from the visualization and analysis modules. For each mipmap tile, we additionally compute min/max values to facilitate hierarchical culling. Segmentation data are processed in a similar way, but in order to avoid interpolating object IDs during down-sampling, we use nearest-neighbor filtering for the computation of segmentation image mipmaps. More elaborate downsampling algorithms (e.g., a rank filter) could also be used. For each tile of the segmentation data, we additionally compute a histogram of all contained object IDs, which facilitates hierarchical culling during the dynamic evaluation of spatial queries.

Table 1. **Object types in *ConnectomeExplorer***. Every query can use an arbitrary combination of these names of object types and pre-defined sets of all known objects of each type. Note that we represent *vesicles* as an attribute of a bouton/synapse instead of as its own object type.

object type / set name	description
cell / cells	cell bodies (somas) of neurons
axon / axons	axons
dendrite / dendrites	dendrites
spine / spines	dendritic spines
bouton / boutons	axon terminals (pre-synaptic); at synapse
synapse / synapses	synaptic clefts between axon and dendrite
psd / psds	post-synaptic densities; indicate synapses
glia / glias	glial cells (these are non-neuronal cells)
– / all	all known objects in a single set

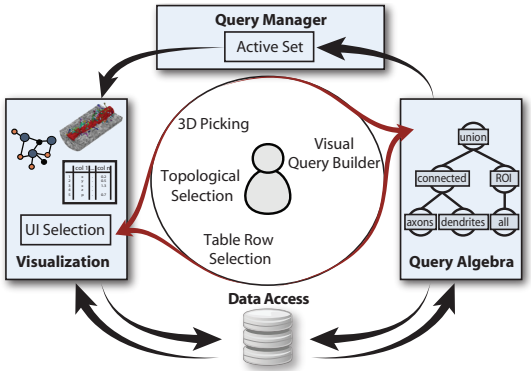


Fig. 3. **User interaction.** A user can dynamically create queries by either using the *visual query builder* or by direct interaction in the provided views via *picking* or *selection*. User input is represented and evaluated via a *query algebra*, and all result sets can be immediately visualized.

The visualization-driven modules of *ConnectomeExplorer* are based on a client/server architecture, where the client is responsible for rendering and evaluation of dynamic queries, while the server is responsible for serving data requests sent by the client. Whenever the server receives a request for a new 3D sub-block that cannot be fulfilled from cached data, it is responsible for constructing that block from the 2D mipmap tiles stored in the visualization archive. The details for this on-the-fly block construction can be found in [22]. Data that has been sent to the client is cached until a pre-defined cache size has been reached. Then it is discarded using a standard LRU scheme.

3.3 Visual Analysis and Exploration

The main goal of *ConnectomeExplorer* is to facilitate the interactive visual exploration and analysis of large volumetric connectomics data sets. A high-level overview of the interaction for analysis is depicted in Fig. 3. Our visualization system is based on a flexible rendering framework that is scalable to petascale data and that supports multiple volumes and linked views. However, the main feature of *ConnectomeExplorer* is that it enables the user to formulate and answer domain-specific questions, either by interactively exploring the data or by posing dynamic queries in an intuitive user interface that translates queries into a query algebra. We support three main types of queries, and their combination: *spatial queries* based on regions of interest or distances, *topological queries* based on neuronal connectivity, and *attribute queries* based on either automatically computed attributes (e.g., the volume of an object), or manually labeled attributes (e.g., the number of vesicles in a synapse). The results of queries are always represented as *sets* (see Sec. 4), which can be examined in all views, used as input for more advanced queries, or stored to and loaded from disk. Statistical parameters can be computed directly on the sets of query results, and can be inspected both visually in histogram and scatterplot views, as well as textually/numerically in table views.

4 Dynamic Knowledge-Based Queries

The main design goal for our dynamic queries was finding the right level of abstraction for our target application, while still providing enough expressive power that enables neuroscientists to answer a wide variety of specific scientific questions. Although at least all attribute-based questions could also be answered via SQL and standard databases [19], we employ a more integrated approach that has similar expressivity, but that is simpler and customized for our requirements by unifying spatial, topological, and attribute queries in an intuitive, knowledge-based manner. Our basic representation for queries is a simple query algebra that is a *set algebra*. That is, all questions can be answered by building on sets of objects, sets of tuples of objects, sets of such sets, and a few simple predicates and operators on sets, which also include computing statistics. See Fig. 4 for a first example.

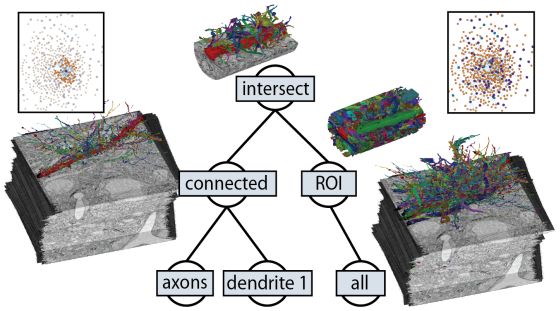


Fig. 4. **Example query tree.** Each user-defined query corresponds to a tree, where the leaves are objects or sets, and the internal nodes are predicates or operators. Each node also corresponds to its result set. Here, the left sub-tree returns all axons connected to dendrite 1 (see connectivity graph in left inset); the right sub-tree evaluates a region of interest. The final result set (tree root) is the set intersection of both.

4.1 Set Algebra for Knowledge-Based Queries

A crucial design criterion of our query algebra was achieving simplicity from a user perspective by allowing the connectivity of objects to be used implicitly, without requiring the user to know what and how the connectivity of different kinds of objects is actually represented and stored internally. In this way, our query algebra enables expressing specific neuroscience questions in a much simpler way than is possible at the abstraction level of standard relational algebra or SQL statements. In contrast to standard database queries, our query algebra allows the connectivity between all kinds of neuronal objects listed in Table 1 to be exploited without having to perform an explicit natural join or a Cartesian product in query languages such as SQL [19]. Explicitly exploiting connectivity information usually requires joining two or more tables on their common attributes, which implies that the user must know what information is stored in which table, if the common attributes specify the desired kind of connectivity, and thus what the result of a natural join will be and what it means. Instead, we allow the direct creation of sets of tuples of objects such that each tuple is only created when the corresponding objects are actually connected.

4.1.1 Basic Concepts

The result of every *query* is an (unordered) *set* $\mathcal{S}$. Each set can contain an arbitrary mixture of (1) individual *objects* $\mathcal{O}_i$, (2) ordered *tuples* $\mathcal{T}_i$ of objects, and also (3) individual sets $\mathcal{S}_i$, i.e., we allow *sets of sets*.

Objects. Each object $\mathcal{O}_i$ has a unique ID, an object type (e.g., *axon*), and named attributes such as *synaptic strength*. All types and attributes are pre-defined (see Table 1). All objects of the same type have the same attributes. The object type itself is also an attribute, which makes querying according to type identical to querying any other attribute.

Sets. A set $\mathcal{S}$ results either from evaluation of a set operator or predicate (Sec. 4.2), or from manual picking/selection in any view of our application (Sec. 5.1.1), in order to create a set that contains the desired objects. A set can contain an arbitrary mixture of objects of different types, and in fact everything could be computed from the pre-defined set **`all`** of all known objects. For convenience, we provide pre-defined sets of all objects of each type, such as the set **`axons`** (all axons), or the set **`dendrites`** (all dendrites). See Table 1. A set can also be the empty set, i.e., $\mathcal{S} = \emptyset$, when a query produces no result.

Tuples. Relationships between objects can be expressed by creating ordered tuples $\mathcal{T}_i =< \mathcal{O}_1, .., \mathcal{O}_n >$. A *set of tuples*, i.e., $\mathcal{S} = \{\mathcal{T}_i\}$, can be created by simply specifying input sets instead of input objects. For example, we denote the creation of a new set of tuples with the (arbitrary) name **`ad`** by **`ad:=<axons,dendrites>`**. This set will contain all axon/dendrite pairs that are *connected*, which is the default behavior. The user also does not need to know or specify how objects are connected, because this knowledge is intrinsic to our system. Our internal representation of the connectivity of neuronal objects knows that axons and dendrites connect by making synapses, and which of them do. Likewise, for other types of objects, different notions of connectivity are employed automatically. If a different kind of connectivity is desired, it can be specified along with the tuple specification. In addition to topological connectivity (Sec. 4.2.2), we allow spatial relationships such as spatial proximity to be used similarly (Sec. 4.2.3).

Our set algebra on purpose does not allow multi-sets, i.e., the example set **`ad`** from above contains only one tuple for each pair of a specific axon and a specific dendrite that are connected, even when they are connected via multiple synapses. However, the user can instead create a set as **`<axons,dendrites,synapses>`**, which then contains a tuple for each known axon/dendrite/synapse triplet.

Sets of sets. In order to group the objects in a set according to some criteria, it can be converted into a set that contains sets, i.e., $\mathcal{S} = \{\mathcal{S}_i\}$, with one $\mathcal{S}_i$ for each group. This is achieved via the **`group`** operator (Sec. 4.2.1) that is often used before computing statistics (Sec. 4.2.5).

4.2 Using Set Predicates and Operators for Queries

In addition to standard set operators (such as set union), projection, and grouping (Sec. 4.2.1), our set algebra unifies three different kinds of queries: (1) *topological* relationships of objects (Sec. 4.2.2), i.e., their connectivity, which is represented internally; (2) *spatial* relationships of objects (Sec. 4.2.3), such as distance or proximity; and (3) queries on *attributes* of objects (Sec. 4.2.4), which can be *physical* properties, e.g., size/volume, or *abstract* properties, e.g., function.

Queries are constructed in an intuitive way by creating new sets via either a (boolean) *predicate* or an *operator*, given one or several existing sets as input. We use the notation **`[predicate]<set>`**, or **`[operator]<set1,set2>`**, etc. for applying a predicate or an operator to a set or to multiple sets, respectively. For example, **`[vesicleCount>50]<synapses>`** creates the set of all synapses with more than 50 vesicles, according to the attribute **`vesicleCount`** of each synapse object in the set **`synapses`**.

Whenever a *predicate* is specified, the result set comprises all elements for which the predicate evaluates to true. For *operators*, the result set depends on the type of operator. Predicates can be restricted to any subset of tuple dimensions for their evaluation.

Each set that is the output of a specific query can be used right away as the input to follow-up queries. In this way, very complex queries can be built up incrementally from simple queries in a hierarchical manner. Fig. 4 illustrates a simple example. Queries can be visualized as trees, where the input sets are the leaves of the tree, and the predicates or operators are the internal nodes of the tree. Sec. 4.3 describes how users can specify queries in a visual way, and how they can visually inspect each result set of the evaluation of a predicate or an operator.

4.2.1 Set operators, projection, and grouping

Set operators. We support the most common operators that operate directly on sets. For convenience, we allow their use in a unary, binary, or *n*-ary fashion where possible. We support the set operators **`union`** (all elements in at least one input set), **`intersect`** (all elements in all input sets), the *relative complement* **`relcomp`** of two sets (all elements in one set, but not the other; i.e., the set difference of two sets), the *absolute complement* **`abscomp`** (all elements in none of the input sets), and the symmetric difference **`diff`** (all elements in exactly one set but not in several). The absolute complement depends on implicit determination of what the enclosing set (*universe*) is, e.g., the complement of a set of axons with respect to the set of all axons.

The projection operator (**`project`**) allows keeping only specified dimensions from tuples, e.g., reducing each tuple with n dimensions to m dimensions ($m < n$), by specifying which m dimensions should be kept. For example, **`project(2)`** projects to the second tuple dimension, **`project(2:4)`** to dimensions 2 to 4.

The grouping operator (**`group`**) converts an input set into a set of sets (one for each group) according to either a common tuple dimension, or a common value—or value range—on a specified attribute.

4.2.2 Topological predicates

We currently support two basic—but very similar—types of topological queries. Both are specified with an identical syntax via the **`connected`** predicate. This predicate determines if objects are either *connected* or *"a part of"*. For example, a connected axon and dendrite, or a spine *of* a dendrite, respectively. Our system encodes and implicitly uses all topological knowledge according to object type.

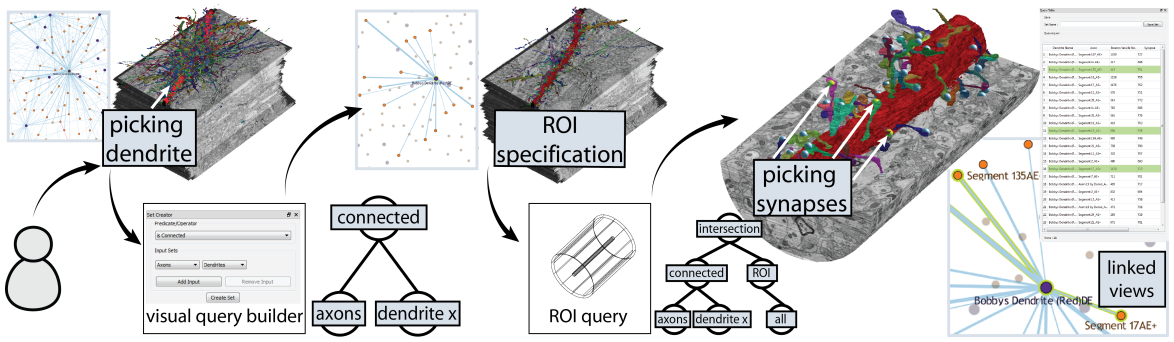


Fig. 5. **Dynamic queries and linked views.** All views allow direct user interactions (e.g., picking, selection) that can be used as input for new queries or for exploration of the data set. New queries are specified in the *visual query builder* and automatically update all views after query evaluation.

4.2.3 Spatial predicates and operators

We support two spatial predicates and one spatial operator:

- The region of interest (**ROI**) *predicate* is true for objects that are in (either fully within, or partially intersecting) a specified ROI, which can be a box (with arbitrary orientation), a sphere, or a cylinder (with arbitrary orientation). For example, **[ROI(***parameters***)]<axons>** results in a set of axons for which the specified ROI predicate is true.
- The **distance** *operator* computes the pair-wise Euclidean distance between objects. The distance between two segmented objects is the distance between the two closest voxels of the two objects. The result set contains a tuple **<obj1,obj2,** *distance* **>** for each tuple (pair) in the input. We allow the optional specification of an accuracy parameter, which enables optimized computation, going from voxel-accurate distance (slowest to compute) to an arbitrary block granularity in voxels (see Sec. 6.2).
- The **distance** (proximity) *predicate* is true depending on a comparison operator on the pair-wise distance between objects, e.g., when objects are closer than a certain threshold. These proximity queries are often much more efficient to evaluate than full distance computations, because they allow exploiting a hierarchical culling scheme to exit early from traversal (see Sec. 6.2).

Like all other queries, spatial queries are evaluated fully dynamically. However, intermediate data required by the implementation (such as blocks of object ID histograms) are cached, which enables potentially re-using them without re-computation when evaluating future queries.

4.2.4 Attribute predicates

We support simple predicates via comparison operators on attributes of objects ($=$, $\neq$, $>$, $<$, etc.). See the example at the beginning of Sec. 4.2 for querying synapses on vesicle count. Attribute queries require the name of the attribute (or object type) that should be used. We enable flexible, knowledge-based queries in the sense that we allow every attribute to be queried on any object type where this is semantically well defined. For example, the attribute *excitatory* internally is an attribute of axon objects and stored only once for each axon. However, it can also be queried on the cell of the axon, each of its synapses, etc.[1] For example, **[excitatory]<synapses>** creates a set of all synapses whose corresponding axon (and cell) is excitatory.

4.2.5 Statistical operators

Statistics can be computed directly by invoking a statistical operator on a set, e.g., counting the number of elements in a set via operator **count**, computing mean, standard deviation, or other basic statistics. All operators except **count** require specifying both the kind of statistic that should be computed, e.g., the *mean* (**avg**), the *standard deviation* (**dev**), as well as the object attribute over which it should be computed. For example, **[avg(vesicleCount)]<synapses>** computes the average number of vesicles of all synapses.

[1]The reason why we currently store this with each axon and not with each cell is that the actual cell bodies can be outside of the imaged tissue block.

Statistics over objects in a set. The corresponding statistic is computed over all objects in the set that have an attribute of the specified name. The result is a tuple like **<'vesicleCount'**, statistic **>**. All objects of the input set that were used to compute this statistic are removed from the set and are altogether replaced by the result tuple. The string 'vesicleCount' in the first tuple dimension can be used exactly like an object attribute name in follow-up queries.

Statistics over sets of sets. For each set in a set, the operator applies itself recursively to that set and replaces it afterward with the result stored in a tuple **<'vesicleCount',** statistic, subsetID **>**. The last dimension is assigned automatically to be unique, and simply serves the purpose of keeping the result tuples of several subsets separate in the enclosing set because we do not allow multi-sets.

Statistics over sets of tuples. Multivariate statistics can be computed automatically over all tuple dimensions, e.g., the covariance matrix or the product-moment correlation coefficients. These computations can be restricted to any subset of dimensions of a set of tuples.

4.3 Visual Query Builder

To support the dynamic creation of queries in a visual way, we have implemented a *visual query builder* (see Fig. 5 and the video). It allows the user to specify an arbitrary number of input objects and sets on which any specified predicate or operator should be evaluated. In addition to using already defined sets as input, the user can also directly select or pick input objects and sets from any of the linked views (see Fig. 5). When the user has finished the specification of inputs and predicates/operators, the *create set* button evaluates the query and creates the corresponding result set. The result set is automatically stored internally, and can be used right away as an input set in subsequent queries. All sets (i.e., default sets and user-created sets) are listed in a *set list* widget, and can be inspected in detail with the *set inspector* widget. The *set inspector* lists all elements of a single set and allows inspecting the actual value of each individual object's attributes.

5 Visualization

The visualization capabilities of *ConnectomeExplorer* comprise several linked views, from a 3D volume view to a topological 2D graph view for neuronal connectivity, a slice view that allows manual segmentation, data and table views for attribute and meta data, and analysis views for visualizing the results of statistics queries. All views are linked and depict the result of queries via the corresponding result set.

5.1 Volume Rendering

Our volume renderer is an extension of previous work [22] that targets scalability to very large electron microscopy volumes. In addition to concurrently rendering segmentation volumes [5], we have extended the 3D volume view to support picking and highlighting objects in 3D, linking to other views, and the automatic visualization of all objects that are in the result set of a query. We employ a data construction back-end that constructs small 3D sub-blocks for a requested position and resolution from 2D image data on demand (see Sec. 3.2). The construction of 3D sub-blocks of data is driven by the visibility of small 3D blocks requested by the ray-caster. Only the blocks that are currently visible are requested and downloaded to GPU memory.

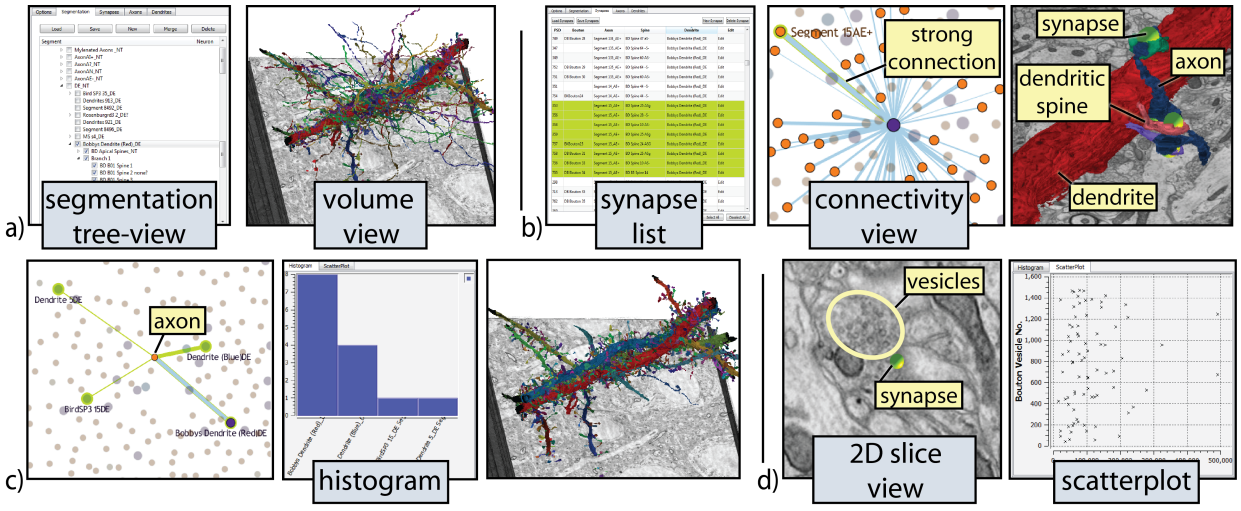


Fig. 6. **Example analysis session.** a) The user starts with a manual inspection of the segmentation, before looking at a single dendrite (in red) and its connected axons. b) Looking at all connections that the red dendrite makes, and querying the axon it makes most synapses with (blue axon). c) Looking at all connections of the blue axon and their strengths, grouped by dendrite, viewed in a histogram. d) Comparing all synapses of the red dendrite based on specific properties: the size of dendritic spines vs. the vesicle count of axons/boutons were analyzed using a scatter plot.

5.1.1 Segmented Data and Synapses

Segmentation data are stored as separate image data, where each voxel stores an integer object ID. In order to accommodate a large number of distinct objects, which is crucial for visualizing the result of automatic EM segmentation algorithms, we currently support up to 24 bits per object ID. This allows representing more than 16 million distinct objects, but could be extended further. Segmentation data are rendered concurrently with the EM data via a multi-volume rendering approach. In addition to the volume data, we can concurrently render synapses at their exact location in volume space, as explained below.

Multi-volume rendering. The modularity of our volume handling and rendering system supports a rather straightforward extension for multi-volume rendering. Our application employs multiple instances of the same virtual memory architecture, and allocates separate cache textures for the EM and segmentation volumes, respectively. Actual mixing of these volumes is performed in the ray-caster. This can be done by using the object ID of the current sample mapped to either a procedurally assigned color, or via a user-defined color table that assigns a specific color to each object ID. Furthermore, we support different rendering modes and individual clipping of each volume. By default, we blend the color of the current sample's object ID with the color of the original EM data after applying a 1D transfer function.

Rendering synapses in 3D. Synapses are rendered as small shaded spheres at their 3D volume position, and combined with the volume rendering using correct visibility compositing. For the latter, we adjust the ray-casting setup step such that rays terminate accordingly when they intersect a synapse. To render a shaded sphere without explicit geometry, we render a view-aligned quad for each synapse sphere, and use a fragment shader to perform shading for those positions that intersect a procedurally computed sphere, and discard the pixels outside.

Picking in 3D. We support picking segmented objects and synapses directly in the 3D volume view. To enable picking objects, the ray-caster makes use of an additional output buffer for object IDs. The ray-casting pass writes out the object ID of the first object that was hit encoded in a 24-bit RGB value. To determine the object at the current mouse position, we then simply do a look-up at the corresponding location in the object ID buffer. Selected objects are automatically highlighted in all linked views, and can be subsequently inspected in more detail, or used as input for a dynamic query. For picking of synapses, we render the synapse IDs into a selection buffer, which we then use to determine the closest, non-occluded synapse at the current mouse position. In order to limit picking to synapses that are currently at least partially visible, we have to restrict which synapses we render into the selection buffer. During ray-casting we therefore write the final depth of each pixel (where a ray leaves the volume or is terminated by early ray termination) into a depth texture. Next, when rendering synapses into the selection buffer, we perform depth testing against this depth texture. This results in restricting the synapses that can be picked to those that are not completely occluded in the volume rendering.

5.1.2 Empty Space Skipping and Culling

We perform empty space skipping depending on both the EM volume and the segmentation volume to speed up rendering and to reduce the amount of GPU memory the ray-caster needs. For EM data, we cull all visible blocks against the current transfer function to determine if a block is fully transparent or not. For segmentation data, we cull against the object IDs that are currently enabled or contained in the current result set. If all objects in a block are disabled or not in the result set, the block is classified as invisible. If a block is classified as invisible, its page table entry is set to empty, no data is downloaded to the GPU, and the block is skipped during ray-casting.

The min/max values and object ID histograms required for culling are initially computed for each 2D image tile in the data-driven part of our pipeline (Sec. 3.2). During the block construction stage, this information is combined according to the requested 3D block, and transmitted to the client. This speeds up empty space skipping and allows culling for spatial queries to be performed hierarchically (Sec. 6).

5.2 Linked Views

Our system offers a variety of views to support users in data analysis and exploration (see Figs. 5 and 6). All views are linked and update to display query results, but also allow for independent data exploration.

5.2.1 2D Connectivity Graph View

We have implemented a graph view to visualize the connectivity between different structures (see Fig. 6b,c). We extract the connectivity information from meta data provided by our collaborators, which is based on tables of annotated structures. In the graph view, nodes correspond to axons or dendrites, and the synapses between them are displayed as links between nodes. The graph layout is computed via a force-directed graph drawing algorithm [18] based on the connectivity of the data. This automatically places nodes with a large number of connections toward the center of the graph, and less-connected nodes toward the edges. The connectivity graph view is fully linked to all other views. It also supports picking, selection, and mouse hovering, to provide user input for queries or to examine individual elements.

5.2.2 2D Segmentation/Slice View

In addition to the 3D volume view, we support 2D slice views. Our slice views employ the same virtual memory architecture as the volume view and support arbitrary slicing planes. Nevertheless, the scien-

tists most often view the *xy*-plane, because of the high in-plane resolution of our EM data. We also provide a manual segmentation tool that allows painting directly in the current slice view. The brush strokes are rendered into an off-screen buffer that is read back, and also transmitted from the client to the server for centralized storage. Our current segmentation tool is completely manual, but in the future we would like to incorporate semi-automatic corrections for proof-reading [37].

5.2.3 Information and Statistics Views

The following views can be used to inspect all object attributes and segmentation information of the data set in more detail.

Segmentation tree view. This view allows flexible hierarchical structuring of the list of segmented objects according to arbitrary user-determined semantics, including which user performed the segmentation. Our collaborating scientists specifically want to be able to subdivide their data into custom categories for visualization in the GUI (see Fig. 6a), which can be independent of the anatomical parent/child relationship of segmented structures. Furthermore, this view also allows quickly enabling/disabling entire sub-trees of objects, assigning different colors to individual elements as well as to entire sub-trees. The latter colors are used when a sub-tree is collapsed to a single node.

Table views. Our system supports displaying meta data of all existing objects. Fig. 6b shows an example of a list of synapses and their properties. Each table supports filtering and sorting based on attributes, is fully linked to all other views, and can either show the result of the current query, or can be used for free manual exploration.

Analysis views. For statistical analysis and comparison tasks, we provide standard views such as histograms and scatterplots, which support dynamic query-based visual analysis (see Fig. 6c,d).

6 Demand-Driven Spatial Query Evaluation

To support the on-the-fly evaluation of spatial queries, such as evaluating a region of interest (ROI) or the proximity of one object to another, we employ an octree-based culling structure that allows quickly finding all volume sub-blocks corresponding to a specified ROI or object.

6.1 Region of Interest (ROI) Evaluation

We currently support the evaluation of cylindrical, spherical, and box-shaped regions of interest. Multiple ROIs can be combined in a query to create more complex ROIs. Once the user has specified a ROI, we compute the set of contained objects in that region, which can either be displayed directly or used as input for subsequent queries.

To enable the demand-driven determination of which segmented objects are inside a ROI, we use a dynamically created octree data structure that stores the list of contained object IDs for each octree node. The octree is only grown and updated on demand. We then traverse the octree to find the biggest nodes that are still completely inside the region of interest, and return the corresponding object IDs. To optionally speed up the computation, two different accuracy levels can be specified. The optimized computation only evaluates nodes that are completely inside a ROI, whereas the accurate computation also evaluates all nodes that intersect the ROI. Only the worst case of intersecting leaf nodes requires iterating over voxels to determine the object IDs inside the ROI. Whenever a volume block is constructed on the server and sent to the client, the histogram of that block is also sent to the client and added to the octree. If during ROI evaluation the histogram of a block is not available yet, it can be requested from the server without having to request the volume data of the entire block.

Finally, to display a ROI in the renderer, we restrict ray-casting to voxels inside the region by simply adjusting the start and stop positions of rays according to the geometry of the specified ROI (see Fig. 5).

6.2 Distance Evaluation

The computation of pair-wise Euclidean distances between objects is based on the octree data structure described above. We base our algorithm on the work of Dyllong and Grimm [15], who compute the distance between two octree-encoded objects at interactive rates, even for octrees with a large number of hierarchy levels. The algorithm is based on the idea to track the nodes that potentially minimize the distance between both objects. To compute the distance between two objects, we update this list iteratively until the set of nodes closest to each other is found. For proximity queries, we can stop this process as soon as two objects are closer/farther than the specified predicate. For voxel-accurate distances, a final step iterates over individual voxels.

Table 2. **Data set statistics** for the data used in Sec. 7. We received two different versions of (partly conflicting) manually labeled synapses, which we then merged into a single consistent data set. Statistics of the segmentation data are given in the middle part of the table. At the bottom we list statistics of three regions of interest that were specified by our collaborators and exhibit a higher density of segmented structures.

data	size			
EM volume	$21,494 \times 25,790 \times 1,850 \times 8$-bit (955 GB)			
segmentation	$10,747 \times 12,895 \times 1,850 \times 24$-bit (716 GB)			
synapses	file 1: 263, file 2: 704, merged file: 943			
object type	**% of vol**	**#objs**	**avg. #voxels/obj**	**avg. # syn**
all	1.22	4108	0.7×10^6	-
axons	0.34	649	1.3×10^6	1.3 (max 14)
dendrites	0.71	104	17.5×10^6	8.9 (max 179)
dend w/spines	0.82	104	20×10^6	8.9 (max 179)
spines	0.11	3055	0.09×10^6	0.1 (max 4)
glia	0.06	23	6.83×10^6	0
cylinder	**% of vol**	**# axons**	**# dendrites**	**# synapses**
cyl 1	0.42	645	104	820
cyl 2	0.10	270	42	189
cyl 3	0.04	120	21	27

7 Results and Discussion

Our collaborating neuroscientists are currently working on segmenting and analyzing an electron microscopy data set of a mouse cortex with a resolution of $21,494 \times 25,790 \times 1,850$ voxels, which we use for the evaluation of our system. Table 2 lists the details of this data set, as well as some of the statistics we have computed via queries. Three different cylindrical regions of interest were specified by our collaborators for queries in accordance with their analysis plans. They have manually segmented several thousand structures over the course of several months, at the voxel level of half the resolution of the EM volume in the *xy*-plane, and are now in the process of automatically generating dense segmentations for selected regions. The current segmentation volume contains over 4,000 labeled objects, including 649 axons, 104 dendrites, several segmented glial cells, and around 3,000 dendritic spines (i.e., small extensions on an excitatory dendrite, where the actual synapses are made). For one cylindrical ROI in the volume (*'cyl 1'*) every structure has been traced. The rest of the volume is only sparsely segmented. Additionally, 943 synapses have been identified and labeled in tables with detailed attributes, including manually counted numbers of vesicles that the corresponding bouton contains.

7.1 Query-Based Analysis Scenarios

We report on several domain-specific questions and analysis tasks for which our domain experts have used *ConnectomeExplorer*. Table 3 summarizes the following scenarios and results, and lists the amount of time our collaborators spent on each individual sub-task.

7.1.1 Axon and Dendrite Analysis

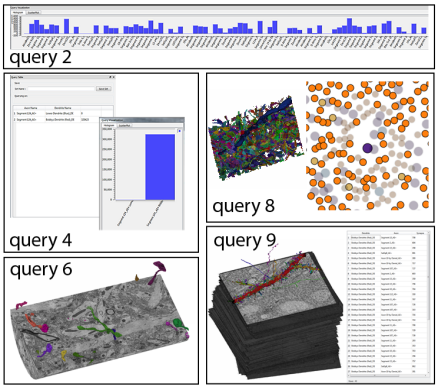
The first questions asked by our domain experts were based directly on attributes of axons, dendrites, and synapses. Pre-synaptic axon terminals (*boutons*) contain neurotransmitters that are essential for transmitting signals over synapses. These neurotransmitters are stored in small membranes called *vesicles*. The number of vesicles has been labeled in our data, because our collaborators are interested in examining the distribution and number of vesicles in different axons and whether there is a correlation between the number of vesicles in a synapse and other attributes of the axon, dendrite, or synapse in question. The specific questions they asked, and the corresponding queries, were:

1. *What is the average number of vesicles of a specific axon that was picked by the user in the 3D view?*

```
1a := [axonId=5]<axons> // sel axon5 from set of axons
1b := <1a, synapses> // tuples axon5 and its synapses
1c := [avg(vesicleCount)]<1b> // avg ves no for axon5
```

Table 3. **Timings and results** for analyzing our use cases. We state the typical time the user took to answer the questions defined in Sec. 7.1. The numbers in the first column refer to the query numbers given there. We also state the time it takes our system to evaluate the query (*query evaluation*), and the time the scientists took to evaluate and look at the results (*post-analysis*). Results of individual queries are shown on the right.

query	user interaction	query evaluation	post-analysis	results
1	exploration/picking: 1 min query builder: 1 min 30 sec	$<$ 1 sec $<$ 2 sec	15 sec	picked 'Segment124' avg vesicle count: 746
2	(2b) query builder: 2 min (2d) query builder: 2 min	$<$ 2 sec $<$ 2 sec	30 sec 15 sec	Subfigure query 2 avg over all its syn: 430.6; only syn with labeled vesicles: 777.4
3	query builder: 45 sec	$<$ 1 sec	15 sec	std deviation: 681.87
4	query builder: 2 min 15 sec	$<$ 3 sec	30 sec	Subfigure query 4
5	exploration/picking: 2 min query builder: 30 sec	$<$ 1 sec $<$ 1 sec	15 sec	*not connected*
6	ROI spec.: 15 sec – 2 min query builder: 1 min 30 sec	data in cache: $<$ 2 sec; else: $<$ 1 min	1 min	Subfigure query 6
7	query builder: 1 min 30 sec	ROI eval + $<$ 2 sec	1 min	see Fig. 4
8	45 sec	$<$ 2 sec	1 min	Subfigure query 8
9	30 sec	$<$ 2 sec	30 sec	Subfigure query 9



2. *What is the average number of vesicles over all axons, and what is the average number of vesicles of each axon?*

```
// set of tuples for all axons and their synapses
2a := <axons,synapses>
2b := [avg(vesicleCount)]<2a> // avg over all axons
2c := [group(1)]<2a> // set of sets, grouped by axon
2d := [avg(vesicleCount)]<2c> // avg for each axon
```

3. *Do all synapses of an axon have the same number of vesicles?*

```
3a := [dev(vesicleCount)]<1b> // std dev of ves count
```

4. *Or does the vesicle number depend on the specific dendrite an axon connects to?*

```
// tuples of connected axons, dendrites, synapses
4a := [axonId=5]<axons, dendrites, synapses>
4b := [group(2)]<4a> // set of sets grouped by dendrite
4c := [dev(vesicleCount)]<4b> // std dev per dendrite
```

Statistical results can be viewed in one of the statistics views (histogram, scatterplot). Our collaborators use this kind of simple analysis to quickly find trends or patterns related to different neuronal object types (e.g., dendrites, synapses) as well as different attributes (e.g., spine size, neurotransmitter type). For dendrites, their main questions are: *What does the size of a spine depend on exactly? Do all spines of a specific dendrite have the same size? Or does the size of a spine depend on the axon it connects to?* These can be answered in the same manner as the axon example above by calculating the average and standard deviation of the spine size at a synapse on a dendrite.

7.1.2 Connectivity Analysis

Connectivity-related questions can be answered via topological predicates, or for simple cases by inspecting the graph view. For example:

1. *Are two structures that are picked in the 3D view connected?*

```
5a := [dendriteId=7]<dendrites> // picked dendrite
5b := [connected]<1a,5a> // empty set if not connected
```

7.1.3 Region of Interest and Spatial Analysis

Our collaborators have completely traced and segmented one cylindrical ROI (*'cyl 1'*). They are therefore particularly interested in queries in this ROI, and the potential similarities and differences of the ROI compared to the entire volume. They would also like to answer high-level questions like *Is the spatial relationship and proximity of a dendrite and an axon related to its connectivity? Does a close proximity through space guarantee a connection?* Their detailed questions were:

1. *What are all the dendrites in the ROI? What are the ones inside the ROI that are smaller than a certain volume?*

```
6a := [ROI(cylinder c)]<dendrites> // dendrites in ROI
6b := [size < 500k]<6a> // small dendrites in ROI
```

2. *Which axons does a specific dendrite connect to in this cylinder?*

```
7a := <5a, axons, synapses> // all connections of dend7
7b := [ROI(cylinder c)]<7a> // dend7 connections in ROI
7c := [project(2)]<7b> // only keep connected axons
```

3. *Which axons in the ROI do* not *connect to the dendrite?*

```
8a := [ROI(cylinder c)]<axons> // axons in ROI
8b := [relcomp]<8a,7c> // ROI axons\connected axons
```

4. *But do they make connections outside the ROI?*

```
9a := [connected]<5a,8b> // all connections
```

7.2 User Evaluation

We describe two specific use cases where our system has helped domain scientists and data analysts to gain new insights into their data. The first user was a neuroscientist interested in analyzing "multiple-hit" axons within a fully segmented region of interest in the volume (Section 7.2.1). The task of the second user was to numerically analyze and clean up the data from segmentation errors (Section 7.2.2).

General feedback from our users was that they are very happy with the key features of our system, such as simultaneous 3D visualization of EM and segmentation data, and a linked graph view for neuronal connectivity. After an initial training period, they were able to use the visual query builder to quickly narrow down their current "working set" to the subset of neuronal structures they were interested in for a certain task. Especially the ability to switch effortlessly and quickly between the different views, defining and extending powerful queries, and interactively selecting structures in the linked views, was seen as very helpful by the scientists. However, they are still at the beginning of their analysis. Many synapses have not been identified and fully labeled yet, and the segmentation is still very sparse in some areas. In the future, more complete data will help them formulate and query more detailed hypotheses that they can analyze with *ConnectomeExplorer*.

7.2.1 Connectivity and Morphology of Multiple-Hit Axons

The goal of our first user was to analyze multiple-hit axons, i.e., axons making multiple synapses with the same dendrite, and to jointly look at their connectivity and morphology. The session started by querying all axons inside ROI *'cyl 1'*. Next, their connections were grouped and counted, to see the dendrites each axon connects to and the number of shared synapses. In the next step, the user navigated to the synapses of these axons in 3D (with a single mouse click) and examined their local morphology. The main goal was to gain intuition for what the data look like and to discover new patterns that would trigger and guide further analysis. In this case, he wanted to analyze the location of the synapses of a multiple-hit axon that are connected to the same dendrite and find out if they are arranged sequentially or if the axon makes synapses to other dendrites in between. A sequential arrangement might indicate that synapses were made by chance, whereas a non-sequential order might indicate an underlying reason for why these two structures are connected. By using our tool the

scientists were able to discover that the location of the synapses was non-sequential, which they now want to analyze in more detail.

After several sessions with *ConnectomeExplorer*, the user found our system extremely helpful for exploring their data. Especially the ability to form and try out new hypotheses interactively was considered to be very useful, because the scientists still know relatively little about their data, simply because they did not previously have adequate tools for analysis. They greatly appreciated and used the graph view for analyzing the connectivity. However, an additional suggestion was that they would like to have a graph view that also represents a simplified spatial morphology of the data instead of only abstract connectivity.

7.2.2 Proof-Reading and Data Clean-Up

The goal of our second user was to proof-read and numerically analyze the segmentation data. *ConnectomeExplorer* can also help in data clean-up tasks performed by data analysts and technical staff, via basic capabilities for adding, deleting, and modifying objects and their properties. The data analyst in our evaluation session used our tool primarily to look at data statistics, such as average voxel size, number of connections, etc. This enabled finding several errors and inconsistencies in the data that he was not aware of before. He found segmentation errors (wrong splits/merges of objects that were visible in the 3D view), incorrectly labeled segmentations (objects labeled as spines but with a huge volume), duplicate synapses (at almost the same location but with different names), and other wrongly assigned attributes. Additionally, our tool enabled identifying "orphans" in the densely segmented ROI *cyl 1*, i.e., objects that only exist inside the cylinder.

User feedback focused on the interactive 3D visualization that was seen as extremely helpful once detailed structures of interest are selected via queries. It was the first time that he could look at a volume rendering of the EM in combination with the segmentation, and to quickly scan for errors and try out hypotheses using the query system. He was also able to perform his numerical analysis. However, in the future he would like our system to incorporate more proximity queries, based on different distance metrics and skeletonized segmentations.

Table 4. **Comparison of visualization systems for connectomics.** Main system goal, targeted data type and size, features (segmentation, 3D view, graph visualization, interactive queries). Abbreviations: *segm.*: *'sa.'*: semi-automatic; *'m.'*: manual; *'S'*: segmentation; *'A'*: annotation. *3D view*: *'2DT'*: textured slice only; *'G'*: geometry; *'VR'*: volume rendering; *'MVR'*: multi-volume rendering. The coloring highlights desired features.

system	data type	max data size	segm.	3D view	graph	queries
Eyewire [1]	*segmentation:* online game-like segmentation					
	EM	sub-cube of 256^3	sa. S	2DT + G	no	no
Neuro Trace [26]	*segmentation:* concurrent 3D vis of volume and segmentation					
	EM	several GB	sa. S	VR	no	no
Mojo [37]	*segmentation:* proof-reading of automatic segmentations					
	EM	< GPU memory	sa. S	no	no	no
RhoANA / Mojo 2.0 [29]	*segmentation:* automatic segmentation and proof-reading					
	EM	GB-TB	a. S	no	no	no
Knossos [24]	*annotation:* rapid reconstruction of neural morphology (skeleton)					
	EM	sub-cube < RAM	m. A	2DT + G	no	no
Catmaid [38]	*annotation:* collaborative annotation of large image stacks					
	EM	> hundreds GB	m. A	G	no	no
Viking Viewer [4]	*annotation:* multi-user annotation of large image stacks					
	EM	> dozens GB	m. A	G	yes	no
Hadwiger et al. [22]	*visualization:* interactive volume rendering of petascale EM data					
	EM	> TB	no	VR	no	no
Rambo3D [2]	*visualization:* visualization of remote annotations					
	EM	sub-cube < GPU	no	2DT	no	no
Connectome Viewer [20]	*processing framework:* python and plugin-based research platform					
	MRI, DTI	variable	plugins	plugins	yes	no
Brain Gazer [9]	*query system:* visual spatial and semantic queries for fly brains					
	confocal	< RAM	no	VR + G	no	yes
Neuron Navigator [32]	*query system:* textual queries for connectivity of fly brains					
	confocal	< GPU memory	no	G	no	yes
Sherbondy et al. [40]	*query system:* dynamic ROI-based connectivity queries					
	MRI, DTI	< 512 MB	no	2DT + G	no	yes
Connectome Explorer (*ours*)	*query system:* query-guided visual analysis of large connectome data					
	EM	> TB	m. S	MVR	yes	yes

7.3 Performance

We have tested our system on quadcore dual-CPU 3.2 GHz machines with 12 GB RAM, and NVIDIA GTX 680 GPU with 2 GB RAM. Optionally, we can run the data server on a different machine that is connected over the network. Detailed timings for 2D mipmap generation, block construction, and volume rendering the EM teravoxel data set used here (12-75 fps in a 1024×768 viewport, depending on the transfer function used) have been reported previously [22].

Multi-volume rendering reduces the performance on average by 10-30%, depending on the transfer function. Our renderer allows the frame rate to be completely decoupled from the time it takes until missing data have been constructed. Block construction for multiple volumes naturally takes longer, because usually twice the amount of data needs to be constructed. Empty space skipping, especially on the sparse segmentation data, results in performance improvements of up to $5\times$, and reduces the amount of cache texture space required.

All queries are evaluated dynamically at run time. Because of the involved data sizes, we do not employ pre-computations. An exception is the computation of min/max values and object ID histograms for each processed 2D mipmap tile, which is done during mipmap generation. Typical query evaluation times are given in Table 3.

7.4 Comparison, Discussion, and Limitations

Our system aims to facilitate the intuitive visual analysis of large-scale connectomics data based on a domain-specific query-based interface. A comparison of our system to other systems is shown in Table 4. We summarize key motivations for our main design decisions, mention alternatives that we have abandoned, and discuss limitations.

3D visualization. The need for 3D visualization of dense, noisy, and highly anisotropic EM data was not obvious to our neuroscientists at the beginning. However, they now see the combination of 3D visualization with powerful query and analysis capabilities as having very large potential for data analysis, especially of the inherent spatial relationships, and for proof-reading segmentations. The design decisions for our volume processing pipeline have been discussed before [22].

Knowledge-based queries. A knowledge-based query algebra geared toward a specific domain has the inherent limitation that it is not as general as a general-purpose query language such as SQL or mathematical analysis tools like Matlab. We use this limitation as an advantage that allows us to greatly simplify the user interaction. However, specifying complex dynamic queries still is not trivial and needs an initial training period to get acquainted with the basic concept.

Visualization of connectivity. Our initial approach for visualizing neuronal connectivity was integrating 2D graph interaction directly in the 3D volume view. After initial evaluation with the domain scientists, we decided for a separate graph view. Only after understanding the abstract connectivity of the data would they now like to include more details by again adding spatial information to the connectivity.

8 Conclusions and Future Work

We have presented *ConnectomeExplorer*, a novel application for the query-guided visual analysis and exploration of large volumetric neuroscience data sets. Our system is based on a powerful query algebra that has already helped our collaborators in connectomics research to answer some of their concrete questions. The variety of queries in conjunction with linked visualization views has allowed them to discover facts about their data set that they did not know before. The queries we enable are made simpler—and at the same time more powerful—by encoding and building on domain-specific knowledge about the intrinsic semantics of the connectivity of neuronal structures.

In the future, we would like to integrate more advanced segmentation and proof-reading methods [29] into our system, and research intuitive 3D navigation metaphors to navigate and explore petascale volume data sets efficiently. Furthermore, we would like to extend our system toward the comparative visualization of query results.

Acknowledgments

We thank Thomas Theußl and Jose Conchello. This project was partially supported by the Intel ISTC-VC, Google, and NVIDIA.

REFERENCES

[1] Eyewire. http://eyewire.org/, 2012. Accessed on 16/06/2013.

[2] Rambo3d. http://openconnectome.github.io/Rambo3D/, 2012. Accessed on 16/06/2013.

[3] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: an implementation and evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 619–626, 1992.

[4] J. Anderson, S. Mohammed, B. Grimm, B. Jones, P. Koshevoy, T. Tasdizen, R. Whitaker, and R. Marc. The viking viewer for connectomics: scalable multi-user annotation and summarization of large volume data sets. *Journal of Microscopy*, 241(1):13–28, 2011.

[5] J. Beyer, M. Hadwiger, A. Al-Awami, W.-K. Jeong, N. Kasthuri, J. W. Lichtman, and H. Pfister. Exploring the connectome: Petascale volume visualization of microscopy data streams. *IEEE Computer Graphics and Applications*, 33(4):50–61, 2013.

[6] 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.

[7] G. Bezgin, A. T. Reid, D. Schubert, and R. Kötter. Matching spatial with ontological brain regions using java tools for visualization, database access, and integrated data analysis. *Neuroinformatics*, 7(1):7–22, 2009.

[8] 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.

[9] S. Bruckner, V. Šoltészová, M. E. Gröller, J. Hladůvka, K. Bühler, J. Yu, and B. Dickson. Braingazer - visual queries for neurobiology research. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Visualization 2009)*, 15(6):1497–1504, Nov. 2009.

[10] W. Cai and G. Sakas. Data intermixing and multi-volume rendering. *Computer Graphics Forum*, 18(3):359–368, 1999.

[11] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8:215–260, 1997.

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

[13] W. de Leeuw, P. Verschure, and R. van Liere. Visualization and analysis of large data collections: a case study applied to confocal microscopy data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1251–1258, 2006.

[14] M. Derthick, J. Kolojejchick, and S. F. Roth. An interactive visual query environment for exploring data. In *Proceedings Symposium on User interface software and technology (UIST)*, pages 189–198, 1997.

[15] E. Dyllong and C. Grimm. A modified reliable distance algorithm for octree-encoded objects. *Proceedings in Applied Mathematics and Mechanics (PAMM)*, 7(1):4010015–4010016, 2007.

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

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

[18] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, Nov. 1991.

[19] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

[20] S. Gerhard, A. Daducci, A. Lemkaddem, R. Meuli, J. Thiran, and P. Hagmann. The connectome viewer toolkit: an open source framework to manage, analyze, and visualize connectomes. *Frontiers in Neuroinformatics*, 5, 2011.

[21] J. Gerken, M. Heilig, H.-C. Jetter, S. Rexhausen, M. Demarmels, W. A. Knig, and H. Reiterer. Lessons learned from the design and evaluation of visual information-seeking systems. *International Journal on Digital Libraries*, 10(2-3):49–66, 2009.

[22] 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.

[23] J. Heer and B. Shneiderman. Interactive dynamics for visual analysis. *Queue*, 10(2):30:30–30:55, Feb. 2012.

[24] M. Helmstädter, K. L. Briggman, and W. Denk. High-accuracy neurite reconstruction for high-throughput neuroanatomy. *Nature Neuroscience*, 14(8):1081–1088, 2011.

[25] V. Jain, B. Bollmann, M. Richardson, D. Berger, M. Helmstädter, K. Briggman, W. Denk, J. Bowden, J. Mendenhall, W. Abraham, K. Harris, N. Kasthuri, K. Hayworth, R. Schalek, J. Tapia, J. Lichtman, and S. Seung. Boundary learning by optimization with topological constraints. In *Proceedings of IEEE CVPR 2010*, pages 2488–2495, 2010.

[26] W.-K. Jeong, J. Beyer, M. Hadwiger, R. Blue, C. Law, A. Vázquez-Reina, C. Reid, J. W. 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.

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

[28] V. Kaynig, T. Fuchs, and J. Buhmann. Neuron geometry extraction by perceptual grouping in sstem images. In *Proceedings of IEEE CVPR 2010*, pages 2902–2909, 2010.

[29] V. Kaynig, A. Vázquez-Reina, S. Knowles-Barley, M. Roberts, T. Jones, N. Kasthuri, E. Miller, J. W. Lichtman, and H. Pfister. Large-scale automatic reconstruction of neuronal processes from electron microscopy images. In *arXiv: 1303.7186 [q-bio.NC]*, 2013.

[30] A. Kuß, S. Prohaska, B. Meyer, J. Rybak, and H.-C. Hege. Ontology-Based Visualization of Hierarchical Neuroanatomical Structures. In *Proceedings of Visual Computing for Biomedicine*, pages 177–184, 2008.

[31] J. W. Lichtman and W. Denk. The big and the small: Challenges of imaging the brain's circuits. *Science*, 334(6056):618–623, 2011.

[32] C.-Y. Lin, K.-L. Tsai, S.-C. Wang, C.-H. Hsieh, H.-M. Chang, and A.-S. Chiang. The neuron navigator: Exploring the information pathway through the neural maze. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium*, pages 35–42, 2011.

[33] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. *Computer Graphics Forum (Proceedings Eurovis)*, 32, 2013.

[34] A. Martin and M. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proceedings IEEE Conference on Visualization 1995*, pages 271–, 1995.

[35] D. Mayerich and J. C. Hart. Volume visualization of serial electron microscopy images using local variance. In *2nd IEEE Symposium on Biological Data Visualization*, pages 9–16, 2012.

[36] D. Osumi-Sutherland, S. Reeve, C. Mungall, F. Neuhaus, A. Ruttenberg, G. Jefferis, and J. Armstrong. A strategy for building neuroanatomy ontologies. *Bioinformatics*, 28(9):1262–1269, 2012.

[37] M. Roberts, W.-K. Jeong, A. Vázquez-Reina, M. Unger, H. Bischof, J. W. Lichtman, and H. Pfister. Neural process reconstruction from sparse user scribbles. In *Proceedings of MICCAI 2011*, pages 621–628, 2011.

[38] S. Saalfeld, A. Cardona, V. Hartenstein, and P. Tomančák. CATMAID: collaborative annotation toolkit for massive amounts of image data. *Bioinformatics*, 25(15):1984–1986, Aug. 2009.

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

[40] A. Sherbondy, D. Akers, R. Mackenzie, R. Dougherty, and B. Wandell. Exploring connectivity of the brain's white matter with dynamic queries. *IEEE Trans. on Vis. and Computer Graphics*, 11(4):419–430, 2005.

[41] B. Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, 1994.

[42] P. R. Smart, A. Russell, D. Braines, Y. Kalfoglou, J. Bao, and N. R. Shadbolt. A visual approach to semantic query design using a web-based graphical query designer. In *Proceedings of the 16th international conference on Knowledge Engineering: Practice and Patterns*, EKAW '08, pages 275–291, Berlin, Heidelberg, 2008. Springer-Verlag.

[43] K. Stockinger, J. Shalf, K. Wu, and E. Bethel. Query-driven visualization of large data sets. In *Proceedings IEEE Visualization 2005*, pages 167–174, 2005.

[44] C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multi-dimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8:52–65, 2002.

[45] Y. Wan, H. Otsuna, C.-B. Chien, and C. Hansen. An interactive visualization tool for multi-channel confocal microscopy data in neurobiology research. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1489–1496, 2009.