Supplementary Method

Software development

FNT is written in C++ with Qt GUI Toolkit[1]. 3D visualization is based on OpenGL API[2]. Tag Image File Format (TIFF), one of the most widely used file format for storing lossless image data, is supported with the help of LibTIFF[3]. FNT is developed under Linux operating system and has been ported to Windows and Mac OS X.

Data preprocessing

Big imaging datasets are split to cubes before tracing. FNT includes an auxil-iary tool, called `slice2cube` that can convert a set of TIFF files (8-bit or 16-bit greyscale) to 3D image files in gzip-compressed[4] Nearly Raw Raster Data (NRRD)[5] format. `slice2cube` uses OpenMP[6] to speed up the conversion when more than one CPU is available.

Data loading

FNT can directly load single small-sized TIFF or NRRD 3D image files. For big data, `slice2cube` generates a text file describing the organization of data. Through this text file, FNT can load a small portion of total data, ~1GB in size, into com-puter memory. Loaded data usually consists of ~8 small neighboring cubes gen-erated by `slice2cube` for smooth transition.

Data used by FNT can be deposited to a local file system or a file server with high-speed network connection. FNT supports fetching remote files with the help of cURL[7] and a wide range of network protocols are covered.

FNT supports datasets with arbitrary number of color channels. Users can flexibly create a multi-channel dataset combining multiple TIFF and NRRD files by writing a simple text file or modifying the one generated by `slice2cube`. At most nine selected channels can be visualized simultaneously and three of them are available for tracing.

Except for imaging data that contains intensity data, additional files containing categorical data such as brain area annotation are also supported. In FNT, users can retrieve the brain area annotation at any position after loading the annotation file.

Visualization

Maximal intensity projection is used to visualize 3D data. 2D visualization of cross-section is also implemented. At most nine selected channels in three groups can be shown at the same time. Each channel is covered by previous channels in the same group when a group is visualized. Images of three groups are then blended together for display on screen. For example, to visualize dual-channel big data, users can select channels of labeled neurons as group 1 and channels of labeled somas as group 2, and ensure that original data channels cover downsampled channels.

Tracing

Reconstructed neurons are commonly represented by a collection of nodes. Each node is associated with properties including type, position, radius and parental node index that can be stored in SWC file format[8]. In FNT, consecutive nodes of a neuron without branching are grouped together and are called an edge. Edges and other related information are saved to files in FNT file format. Conversion between FNT format and SWC format is supported.

FNT tracing process consists of three steps: finding a putative path, examining the putative path and extending the current neurite tree. Starting from a current position, one can choose a target position by mouse clicks and find a putative connection between them. This process of connecting two positions has been used in other tracing tools[9]. In FNT, we included a new operation: finding a putative path only with a starting position. Without picking another position, this operation allows a putative path to be found with a single key stroke and therefore is much faster than the previous method that requires mouse clicks for picking another position. The plugin interface allows different algorithms to be used for finding and refining putative paths.

If the putative path is acceptable, it can be added to the current neurite tree with a key stroke. A new edge will be created when necessary. If it is considered unacceptable, users can just ignore it and try to find another putative path. In FNT, users' confirmation at each step ensures the correctness of tracing.

At a branch point, users can create an initial edge for each new branch and continue tracing one of them. Once the current branch has been traced to the end, it will be marked as finished with a key stroke. FNT keeps the records of the end points of all unfinished branches and will automatically jump to the nearest unfinished end point when the current branch is finished. A neuron tree can be traversed in this way until it is completed.

Algorithms for finding putative paths

FNT allows different algorithms to be used for finding putative paths. We currently implemented three simple algorithms for tracing based on previous methods with minor modifications.

The path connecting two positions is found using A* algorithm[10]. Given a graph and two vertices in this graph, A* algorithm tries to find the shortest path connecting the two vertices. Here each voxel is a vertex and neighboring voxels are connected. Distance between two neighboring voxels i and j is

$$\mathrm{d}(i,j) = \frac{\mathrm{d}^{\mathrm{E}}(i,j)}{2(V_i + v_0)} + \frac{\mathrm{d}^{\mathrm{E}}(i,j)}{2(V_j + v_0)},$$

where $\mathrm{d}^{\mathrm{E}}(i,j)$ is the Euclidean distance between centers of voxel i and j, V_k is the intensity value at voxel k and $v_0 > 0$ is a tunable parameter. This definition of distance takes into account both Euclidean distance and voxel intensity along the path. The shortest path from seed voxel s to target voxel t is found by extending existing partial paths starting from s. At each iteration, the partial path with shortest estimated overall distance is extended. Overall distance of partial path $(s, i_1, i_2, \cdots, i_n)$ is estimated with $\mathrm{f}(s, i_1, i_2, \cdots, i_n) + \mathrm{h}(i_n, t)$. $\mathrm{f}(s, i_1, i_2, \cdots, i_n)$ is the total distance of the partial path:

$$\mathrm{f}(s, i_1, i_2, \cdots, i_n) = \mathrm{d}(s, i_1) + \mathrm{d}(i_1, i_2) + \cdots + \mathrm{d}(i_{n-1}, i_n)$$

The heuristic function $\mathrm{h}(i_n, t)$ estimates the shortest distance from i_n to t:

$$\mathrm{h}(i_n, t) = \frac{\mathrm{d}^{\mathrm{E}}(i_n, t)}{(V_s + \sum_{1 \le k \le n} V_{i_k})/(n+1) + v_0}.$$

To find a putative path from current position, without a specific target position, we used Dijkstra's algorithm[11], which is a special case of A* algorithm with $\mathrm{h}(i,t) = 0$. The distance function used here is slightly different from $\mathrm{d}(i,j)$ to

avoid harsh curvatures. Each considered voxel i is associated with a vector $\mathrm{T}(i)$. For the seed voxel, the associated vector is a unit vector tangential to the current edge at the current position, or zero vector if the current position is not on an edge. For other voxels, the vector is estimated from the positions of their ancestor voxels. The distance function is then calculated with

$$\mathrm{r}(i,j) = \left| \frac{P_j - P_i}{2|P_j - P_i|} + \frac{\mathrm{T}(i)}{2} \right|$$

$$\mathrm{d}'(i,j) = \frac{\mathrm{d}^{\mathrm{E}}(i,j)}{2(\mathrm{r}(i,j)V_i + v_0)} + \frac{\mathrm{d}^{\mathrm{E}}(i,j)}{2(\mathrm{r}(i,j)V_j + v_0)},$$

where P_i is the position of voxel i and $|P|$ is the length of vector P.

Putative paths are processed to reduce the number of nodes using Visvalingam's algorithm[12]. To estimate the radii of nodes in putative paths, we used the Rayburst[13] method. Intensity values on 16 equally distributed rays starting from the node, perpendicular to the direction of the edge at the node, are sampled. For each ray the distance at which intensity value decreases the most is determined. The mean of the 2-nd to the 8-th shortest distances is used as an estimate of the radius.

Clustering of single-neuron projections by FNT-dist

To cluster single-neuron projections, we defined distance or dissimilarity between two neurons based on their tracing results represented by two trees of nodes. The distance between two neurons is calculated based on their alignment, which is defined by the mapping of each node in one neuron to a node in the other neuron. We used dynamic programming to find the best alignment that minimizes the overall Euclidean distance between each node and its mapped node.

Let N_i be the i-th node of a neuron N with $|N|$ nodes. The first node N_1 is chosen as the root node. N_i^* denotes the parent of N_i and $N_1^* = N_1$. $|N_i|$ is the number of child nodes of N_i and $N_{i,j}$ is the j-th child node of N_i for $1 \leq j \leq |N_i|$. $\mathrm{T}(N_i)$ is the sub-tree rooted at N_i. Define $\mathrm{F}(N_i)$ as the set of sub-trees $\{\mathrm{T}(N_{i,j}) | 1 \leq j \leq |N_i|\}$. In the case $|N_i| = 0$, $\mathrm{F}(N_i)$ is the empty set Φ.

Next we search for the best alignment between neurons N and M by calculating value $\mathrm{P}[\mathrm{T}(N_1), \mathrm{T}(M_1)]$, the minimum penalty for mapping nodes of neuron N to nodes in neuron M. The penalty of mapping N_i to M_j is the weighted distance between segment $\overline{N_i N_i^*}$ and N_j:

$$\mathrm{Pn}[N_i, M_j] = |N_i - N_i^*| \cdot \min_{0 \le t \le 1} |[N_i^* + t(N_i - N_i^*)] - M_j|,$$

where $|A - B|$ is the Euclidean distance between A and B. The total penalty of mapping N_i and it's descendant nodes to M_j is defined recursively:

$$\mathrm{Pm}[\mathrm{T}(N_i), M_j] = \mathrm{Pn}[N_i, M_j] + \sum_{1 \le k \le |N_i|} \mathrm{Pm}[\mathrm{T}(N_{i,k}), M_j].$$

If $|N_i| = 1$ and $|M_j| = 1$, $\mathrm{P}[\mathrm{T}(N_i), \mathrm{T}(M_j)]$ is

$$\min \begin{cases} \mathrm{Pm}[\mathrm{T}(N_i), M_j^*] + \mathrm{Pm}[\mathrm{T}(M_j), N_i^*] \\ \mathrm{Pn}[N_i, M_j^*] + \mathrm{P}[\mathrm{T}(N_{i,1}), \mathrm{T}(M_j)] \\ \mathrm{Pn}[M_j, N_i^*] + \mathrm{P}[\mathrm{T}(N_i), \mathrm{T}(M_{j,1})] \end{cases}.$$

If $|N_i| = 1$ and $|M_j| \neq 1$, $\mathrm{P}[\mathrm{T}(N_i), \mathrm{T}(M_j)]$ is

$$\min \begin{cases} \mathrm{Pm}[\mathrm{T}(N_i), M_j^*] + \mathrm{Pm}[\mathrm{T}(M_j), N_i^*] \\ \mathrm{Pn}[N_i, M_j^*] + \mathrm{P}[\mathrm{T}(N_{i,1}), \mathrm{T}(M_j)] \\ \mathrm{Pn}[M_j, N_i^*] + \min_{1 \le k \le |M_j|} \mathrm{P}[\mathrm{T}(N_i), \mathrm{T}(M_{j,k})] \\ \qquad + \sum_{1 \le l \le |M_j|, l \neq k} \mathrm{Pm}[\mathrm{T}(M_{j,l}), N_i^*] \end{cases}.$$

The score is calculated similarly for cases where $|N_i| \neq 1$ and $|M_j| = 1$.

For $F_1 \subseteq \mathrm{F}(N_i)$ and $F_2 \subseteq \mathrm{F}(M_j)$, define $\mathrm{Pf}[(N_i, F_1), (M_j, F_2)]$ as

$$\begin{cases} \min_{A \in F_1, B \in F_2} \mathrm{P}[A, B] & \\ \qquad + \mathrm{Pf}[(N_i, F_1 - \{A\}), (M_j, F_2 - \{B\})] & \text{if } F_1 \neq \Phi \text{ and } F_2 \neq \Phi \\ \sum_{A \in F_1} \mathrm{Pm}[A, M_j] + \sum_{B \in F_2} \mathrm{Pm}[B, N_i] & \text{otherwise} \end{cases}.$$

Then if $|N_i| \neq 1$ and $|M_j| \neq 1$, $\mathrm{P}[\mathrm{T}(N_i), \mathrm{T}(M_j)]$ is

$$\min \begin{cases} \mathrm{Pm}[\mathrm{T}(N_i), M_j^*] + \mathrm{Pm}[\mathrm{T}(M_j), N_i^*] \\ \mathrm{Pn}[N_i, M_j^*] + \min_{1 \le k \le |N_i|} \mathrm{P}[\mathrm{T}(N_{i,k}), \mathrm{T}(M_j)] \\ \qquad + \sum_{1 \le l \le |N_i|, l \neq k} \mathrm{Pm}[\mathrm{T}(N_{i,l}), M_j^*] \\ \mathrm{Pn}[N_i, M_j^*] + \mathrm{Pn}[M_j, N_i] + \mathrm{Pf}[(N_i, \mathrm{F}(N_i)), (M_j, \mathrm{F}(M_j))] \\ \mathrm{Pn}[M_j, N_i^*] + \min_{1 \le k \le |M_j|} \mathrm{P}[\mathrm{T}(M_{j,k}), \mathrm{T}(N_i)] \\ \qquad + \sum_{1 \le l \le |M_j|, l \neq k} \mathrm{Pm}[\mathrm{T}(M_{j,l}), N_i^*] \\ \mathrm{Pn}[M_j, N_i^*] + \mathrm{Pn}[N_i, M_j] + \mathrm{Pf}[(N_i, \mathrm{F}(N_i)), (M_j, \mathrm{F}(M_j))] \end{cases}.$$

$P[T(N_1), T(M_1)]$ is the lowest penalty of mapping nodes between neuron N and M. With backtracing, all $Pn[N_i, M_j,]$ and $Pn[M_k, N_l,]$ terms in $P[T(N_1), T(M_1)]$ can be extracted. The set of these terms define the alignment of neuron N and M. An example of aligning two neurons is shown in **Supplementary Figure 1**.

Based on this alignment, various dissimilarity measures can be defined. Here we use $P[N_1, M_1]/(L(N)+L(M))$ as the distance between neuron N and M, where $L(N)$ is the total length of neuron N.

References

1. *Qt | Cross-platform software development for embedded & desktop*. (2020). *Qt | Cross-platform software development for embedded & desktop*. Retrieved from https://www.qt.io/

2. *OpenGL — The Industry Standard for High Performance Graphics*. (2020). *OpenGL — The Industry Standard for High Performance Graphics*. Retrieved from https://www.opengl.org/

3. *LibTIFF — TIFF Library and Utilities*. (2020). *LibTIFF — TIFF Library and Utilities*. Retrieved from http://libtiff.maptools.org/

4. *A Massively Spiffy Yet Delicately Unobtrusive Compression Library*. (2020). *A Massively Spiffy Yet Delicately Unobtrusive Compression Library*. Retrieved from http://zlib.net/

5. *Definition of NRRD File Format*. (2020). *Definition of NRRD File Format*. Retrieved from http://teem.sourceforge.net/nrrd/format.html

6. *OpenMP*. (2020). *OpenMP*. Retrieved from http://www.openmp.org/

7. *Libcurl - the multiprotocol file transfer library*. (2020). *Libcurl - the multiprotocol file transfer library*. Retrieved from https://curl.haxx.se/libcurl/

8. Cannon, R. C., Turner, D. A., Pyapali, G. K., & Wheal, H. V. (1998b). An online archive of reconstructed hippocampal neurons. *Journal of Neuroscience Methods*, *84*(1-2), 49–54.

9. Longair, M., Baker, D. A., & Armstrong, J. D. (2011). Simple Neurite Tracer: open source software for reconstruction, visualization and analysis of neuronal processes.. *Bioinformatics*, *27*(17), 2453–2454.

10. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107. doi:10.1109/TSSC.1968.300136

11. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271. doi:10.1007/BF01386390

12. Visvalingam, M. & Whyatt, J. D. (1993). Line generalisation by repeated elimination of points. *The Cartographic Journal*, *30*(1), 46-51. doi:10.1179/000870493786962263

13. Rodriguez, A., Ehlenberger, D. B., Hof, P. R., & Wearne, S. L. (2006). Rayburst sampling, an algorithm for automated three-dimensional shape analysis from laser scanning microscopy images. *Nat. Protocols*, *1*(4), 2152–2161. doi:10.1038/nprot.2006.313