Supplementary Information: Flexible Learning-Free Segmentation and Reconstruction for Sparse Neuronal Circuit Tracing

Ali Shahbazi\textsuperscript{1,+}, Jeffery Kinnison\textsuperscript{1,2,+}, Rafael Vescovi\textsuperscript{2,3}, Ming Du\textsuperscript{4}, Robert Hill\textsuperscript{5}, Maximilian Jösch\textsuperscript{6}, Marc Takeno\textsuperscript{7}, Hongkui Zeng\textsuperscript{7}, Nuno Maçarico da Costa\textsuperscript{7}, Jaime Grutzendler\textsuperscript{5}, Narayanan Kasthuri\textsuperscript{2,3}, and Walter J. Scheirer\textsuperscript{1,*}

\textsuperscript{1}Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA
\textsuperscript{2}Center for Nanoscale Materials, Argonne National Laboratory, Lemont, IL, USA
\textsuperscript{3}Department of Neurobiology, University of Chicago, Chicago, IL, USA
\textsuperscript{4}Department of Materials Science and Engineering, Northwestern University, Evanston, IL, USA
\textsuperscript{5}Department of Neurology, Yale University, New Haven, CT, USA
\textsuperscript{6}Neuroethology Group, IST Austria, Klosterneuburg, Austria
\textsuperscript{7}Allen Institute for Brain Science, Seattle, WA, USA
\textsuperscript{*}walter.scheirer@nd.edu
\textsuperscript{+}These authors contributed equally to this work.

ABSTRACT

This document contains all supplemental figures and methods descriptions.

Supplementary Methods: Evaluation Metrics

In our experiments we compared our EM reconstructions and one of our \(\mu\)CT X-ray stacks against expert annotated ground-truth. Since 3D reconstruction is the goal of the work, we chose Hausdorff distance as an evaluation tool. It is a well-known evaluation method for 3D meshes and reconstructions\textsuperscript{1}. Instead of one-by-one matching as is done when calculating variation of information, which checks the location of each point on the first model with a similar location on second model, Hausdorff distance considers many-to-many correspondence. Consider a 3D reconstruction \(A\) and corresponding 3D ground-truth \(B\):

\[
A = \{a_1, a_2, \ldots, a_n\} \quad \text{and} \quad B = \{b_1, b_2, \ldots, b_n\}.
\]

The Hausdorff distance from \(A\) to \(B\) will be \(\hat{\delta}_H(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\|\).

Throughout this article, we use the convention \(A\) is \(1/1000\) of the distance of \(B\). Thus when we say \(A\) is in \(1/1000\) of the distance of \(B\), this means the average distance between \(A\) and \(B\) is less than 0.001.

Supplementary Methods: 3D U-Net Training

For \(\mu\)CT X-ray segmentation, a standard 3D U-Net model\textsuperscript{2} was trained on the manual segmentation of the SRB dataset\textsuperscript{3}, which consists of a 300 \(\times\) 300 \(\times\) 100 voxel volume. Due to memory constraints, each training batch consisted of a single 256 \(\times\) 256 \(\times\) 19 block augmented by randomized rotation, transpose, and grayscale intensity scaling. The model was trained for 60 epochs using weighted mean-squared error loss and the Adam optimizer\textsuperscript{4} with a learning rate of \(1 \times 10^{-4}\). Training for \(sSEM\) APEX2-positive SAC segmentation proceeded using the same protocol, however the model was trained on manual segmentations created by Jösch \textit{et al.}\textsuperscript{5}.

Supplementary Table S 1. Running time comparison to the previous version of the pipeline\textsuperscript{5}.

<table>
<thead>
<tr>
<th>Method</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jösch \textit{et al.}\textsuperscript{5}</td>
<td>136 min</td>
</tr>
<tr>
<td>2D FLoRIN</td>
<td>24.82 min</td>
</tr>
<tr>
<td>3D U-Net</td>
<td>3559.3 min</td>
</tr>
</tbody>
</table>
**Supplementary Figure S 1.** The 2D Identification process in FLoRIN first registers all connected components discovered in the Identification stage into a single plane. Connections are preserved by connecting components that overlap with one another in 2D, and unconnected components are discarded. The remaining components are then placed into a 3D volume.

**Supplementary Figure S 2.** A comparison of the performance of global versus local thresholding methods on a noisy image. (Left) Neural images are often subject to large shifts in the grayscale distribution that can hamper reconstruction efforts. (Center) Otsu’s method\(^6\) binarizes images using a global threshold value, however due to grayscale shifts the binarization includes large portions of the image background. (Right) Our LAT algorithm operates by observing a local neighborhood around each voxel to reduce the impact of distant noise. In this case, LAT captures more of the features of interest despite large grayscale shifts.

**References**

Supplementary Figure S 3. A comparison of SRB (A) manual annotations, (B) 3D U-Net cell segmentation, and (C) 3D FLoRIN cell segmentation. In general, 3D U-Net tends to over-segment the cells, incurring merge errors with nearby cells and on correctly identifying a vasculature segment as a cell. 3D FLoRIN, on the other hand, clearly separates grouped cells and does not misclassify vasculature.
**Data:** img: an n-dimensional image to threshold  
**Data:** n: the dimensionality of img  
**Data:** d: an n-tuple containing the size of each dimension of img  
**Data:** s: an n-tuple containing the dimensions of the box around each pixel  
**Data:** t: the threshold value to use, number in range [0, 1]  
**Result:** binarization of img

let out be an array the same size as img;  
intImg = img;  
for i in 1..n do  
  // Compute the cumulative summation over dimension i  
  intImg = cumulativeSummation(intImg, i);  
end

let low, hi, vertex be length n arrays filled with zeros;  
parity = n mod 2;  
foreach element e in intImg do  
  x = index(intImg, e);  
  for i in 1..n do  
    low[i] = x[i] - s[i] / 2;  
    hi[i] = x[i] - s[i] / 2;  
    if low[i] < 1 then  
      low[i] = 0;  
    end  
    if hi[i] > d[i] then  
      hi[i] = d[i];  
    end  
  end
  count = Π^n_i=1 hi[i] - low[i];  
  sum = 0;  
  foreach idx in indices do  
    p = 0;  
    for i in 1..n do  
      p = p + idx[i];  
      if idx[i] = 1 then  
        vertex[i] = hi[i];  
      else  
        vertex[i] = low[i];  
      end  
    end
    p = p mod 2;  
    if p = parity then  
      sum = sum + intImg[vertex];  
    else  
      sum = sum - intImg[vertex];  
    end
  end
  if img[x] × count ≤ sum × (1.0 - t) then  
    out[x] = 0;  
  else  
    out[x] = 1;  
  end
end
return out

**Algorithm 1:** Local Adaptive Thresholding