NEUROSCIENCE IN THE DATABASE

MAPPING THE MORPHOLOGY OF NEURONS AND SYNAPSES WITH CATMAID AND POSTGIS

Tom Kazimiers | kazmos GmbH

PostGIS Day 2021, November 18

WHO?

- Tom Kazimiers, Main developer of CATMAID, MSc CS (Dipl.Inf.) TU Dresden
- 3+ years at MPI CBG (Dresden, DE)
- 6+ years at HHMI Janelia Research Campus (Ashburn, Virginia, USA)
- Since 2020: Open Source Research Software Engineer and founder of consultancy kazmos GmbH (DE)

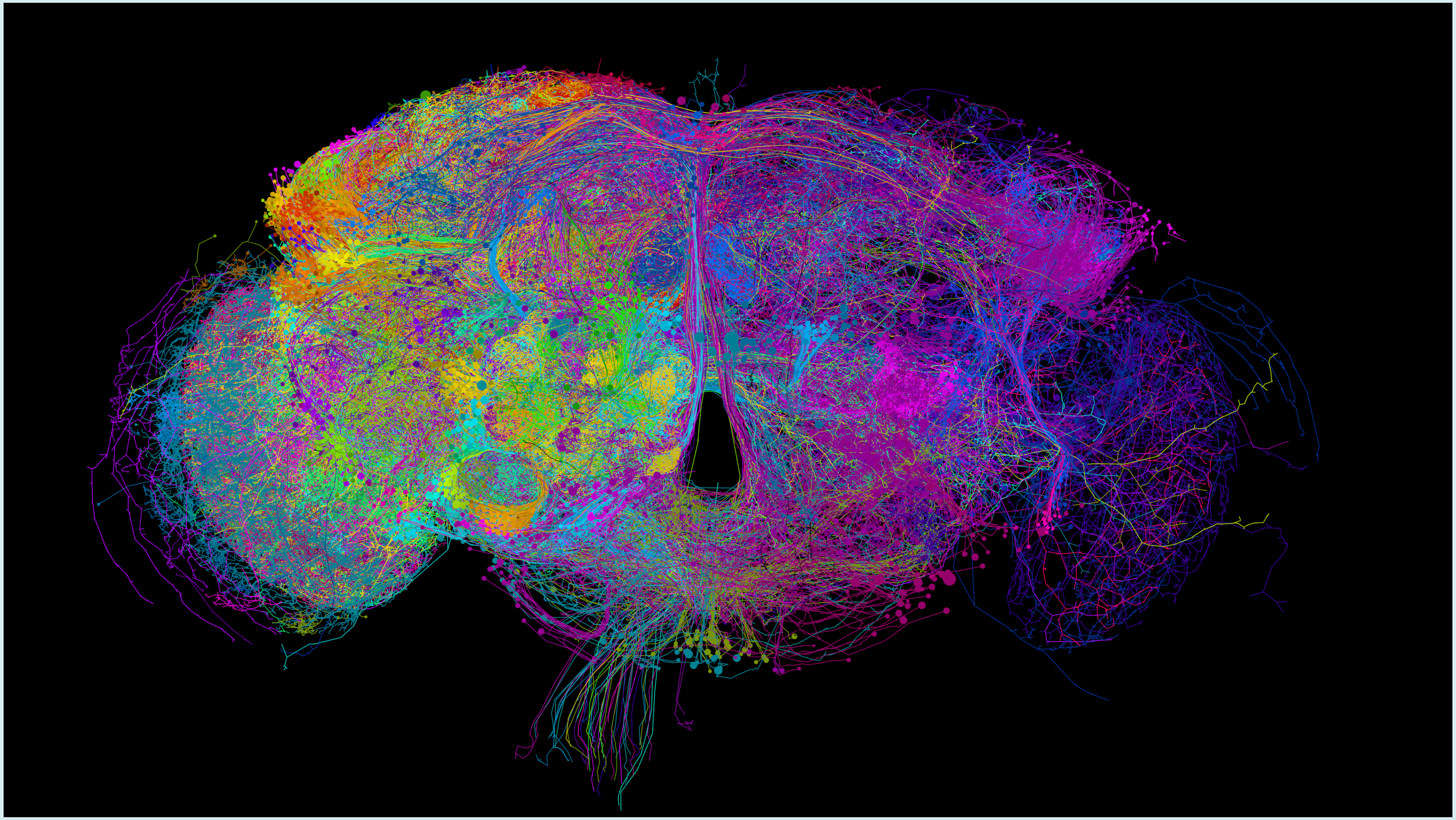
TOPICS

- Motivation and data
- Overview
- Data representation
- Neuron graph queries
- PostGIS queries

MOTIVATION: CONNECTOMICS

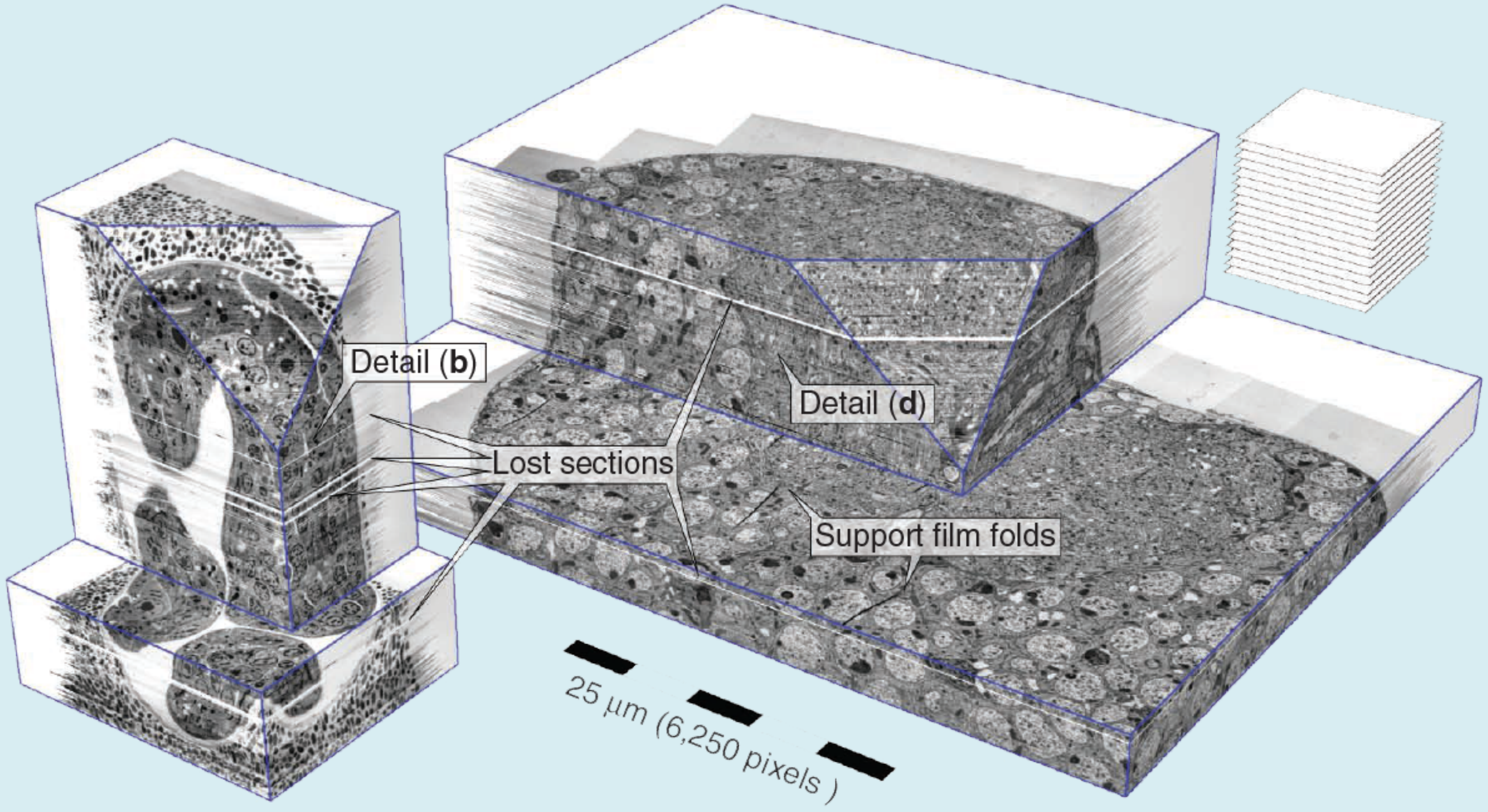
- Structure/development of central nervous system
- Function of specific neurons and neuron classes
- Synapses and resulting networks
- How is behavior controlled?
- Fundamental research

Model organism *Drosophila Melanogaster* ("fruit fly")
Images: Wikimedia, CC-BY-SA 2.5 (left) and CC-BY-SA 4.0 (right)



~7 meters of neurons in brain of fruit fly, 20 labs, 186 users
created and reviewed 33 million nodes in more than 60 person years
Image by Philipp Schlegel, Jefferis lab

IMAGE DATA: ELECTRON MICROSCOPY



ssTEM: Elastic volume reconstruction from series of ultra-thin microscopy sections
S. Saalfeld, R. Fetter, A. Cardona, *et al.*, *Nature Methods*, 2012

REPRESENTATIVE DATA SETS

- **Drosophila Larva L1 Dataset (ssTEM)**:
  28,128 x 31,840 x 4,841 px @ 3.8 x 3.8 x 50 nm/px
  0.9TB as JPEG tiles (512 x 512)
- **Drosophila FAFB Dataset (ssTEM)**:
  253,952 x 155,648 x 7,063 px @ 4 x 4 x 40 nm/px
  10.9TB as JPEG tiles (1024 x 1024)
- **Drosophila Hemibrain Dataset (FIBSEM)**:
  40,959 x 34,815 x 43,007 px @ 8 x 8 x 8 nm/px
  5.3TB as Neuroglancer Precomputed (64 x 64 x 64)

- **Compare: OpenStreetMap tiles (z18)**:
  Scandinavian countries: 1TB (Geofabrik)
  North + South America: 11TB (Geofabrik)
  North America: 5.4TB (Geofabrik)

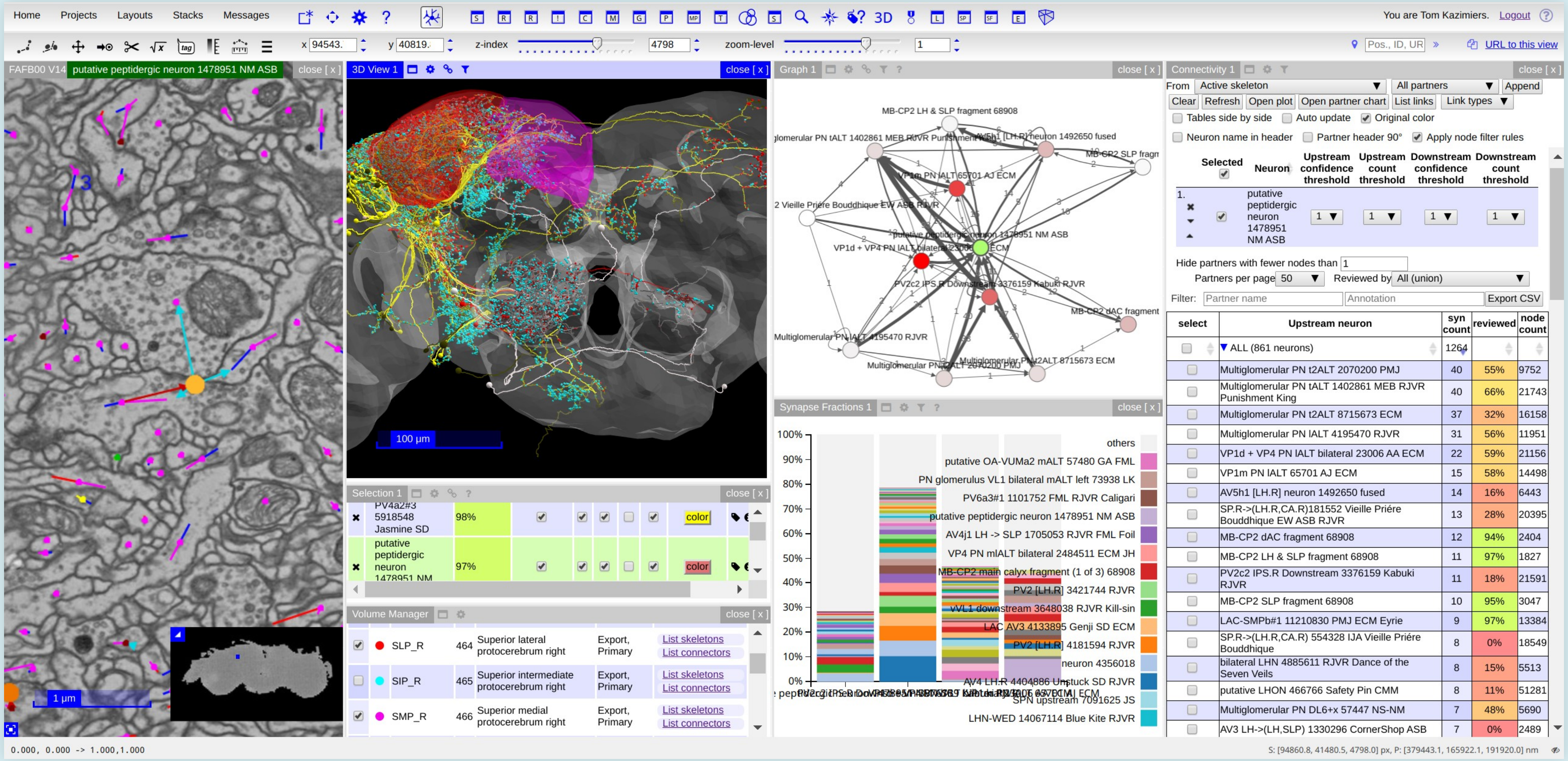
CATMAID

Collaborative Annotation Toolkit

for

Massive Amounts of Image Data

USER INTERFACE I



Home
Projects
Layouts
Stacks
Messages
3D
You are Tom Kazimiers. Logout
x 94543.
y 40819.
z-index
4798
zoom-level
1
Pos., ID, UR
URL to this view
FAFB00 V14
putative peptidergic neuron 1478951 NM ASB
close [x]
3D View 1
Graph 1
Connectivity 1
1 µm
100 µm
MB-CP2 LH & SLP fragment 68908
Multiglomerular PN tALT 1402861 MEB RJVR Punishment King
AV5h1 [LH.R] neuron 1492650 fused
MB-CP2 SLP fragment
VP1m PN lALT 65701 AJ ECM
SP.R->(LH.R,CA.R)181552 Vieille Priére Bouddhique EW ASB RJVR
putative peptidergic neuron 1478951 NM ASB
VP1d + VP4 PN lALT bilateral 23006 AA ECM
PV2c2 IPS.R Downstream 3376159 Kabuki RJVR
MB-CP2 dAC fragment
Multiglomerular PN lALT 4195470 RJVR
Multiglomerular PN t2ALT 8715673 ECM
Multiglomerular PN t2ALT 2070200 PMJ
Synapse Fractions 1
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
others
putative OA-VUMa2 mALT 57480 GA FML
PN glomerulus VL1 bilateral mALT left 73938 LK
PV6a3#1 1101752 FML RJVR Caligari
putative peptidergic neuron 1478951 NM ASB
AV4j1 LH -> SLP 1705053 RJVR FML Foil
VP4 PN mlALT bilateral 2484511 ECM JH
MB-CP2 main calyx fragment (1 of 3) 68908
PV2 [LH.R] 3421744 RJVR
VL1 downstream 3648038 RJVR Kill-sin
LAC AV3 4133895 Genji SD ECM
PV2 [LH.R] 4181594 RJVR
neuron 4356018
AV4 LH.R 4404886 Unstuck SD RJVR
SPN upstream 7091625 JS
LHN-WED 14067114 Blue Kite RJVR
Selection 1
Volume Manager

Connectivity 1

From: Active skeleton ▼ | All partners ▼ | Append

Clear | Refresh | Open plot | Open partner chart | List links | Link types ▼

☐ Tables side by side ☐ Auto update ☑ Original color

☐ Neuron name in header ☐ Partner header 90° ☑ Apply node filter rules

	Selected	Neuron	Upstream confidence threshold	Upstream count threshold	Downstream confidence threshold	Downstream count threshold
1.	☑	putative peptidergic neuron 1478951 NM ASB	1 ▼	1 ▼	1 ▼	1 ▼

Hide partners with fewer nodes than 1

Partners per page 50 ▼ Reviewed by All (union) ▼

Filter: Partner name | Annotation | Export CSV

select	Upstream neuron	syn count	reviewed	node count
☐	▼ ALL (861 neurons)	1264		
☐	Multiglomerular PN t2ALT 2070200 PMJ	40	55%	9752
☐	Multiglomerular PN tALT 1402861 MEB RJVR Punishment King	40	66%	21743
☐	Multiglomerular PN t2ALT 8715673 ECM	37	32%	16158
☐	Multiglomerular PN lALT 4195470 RJVR	31	56%	11951
☐	VP1d + VP4 PN lALT bilateral 23006 AA ECM	22	59%	21156
☐	VP1m PN lALT 65701 AJ ECM	15	58%	14498
☐	AV5h1 [LH.R] neuron 1492650 fused	14	16%	6443
☐	SP.R->(LH.R,CA.R)181552 Vieille Priére Bouddhique EW ASB RJVR	13	28%	20395
☐	MB-CP2 dAC fragment 68908	12	94%	2404
☐	MB-CP2 LH & SLP fragment 68908	11	97%	1827
☐	PV2c2 IPS.R Downstream 3376159 Kabuki RJVR	11	18%	21591
☐	MB-CP2 SLP fragment 68908	10	95%	3047
☐	LAC-SMPb#1 11210830 PMJ ECM Eyrie	9	97%	13384
☐	SP.R->(LH.R,CA.R) 554328 IJA Vieille Priére Bouddhique	8	0%	18549
☐	bilateral LHN 4885611 RJVR Dance of the Seven Veils	8	15%	5513
☐	putative LHON 466766 Safety Pin CMM	8	11%	51281
☐	Multiglomerular PN DL6+x 57447 NS-NM	7	48%	5690
☐	AV3 LH->(LH,SLP) 1330296 CornerShop ASB	7	0%	2489

Selection 1

✖	PV4a2#3 5918548 Jasmine SD	98%	✓	✓	✓	☐	✓	color
✖	putative peptidergic neuron 1478951 NM	97%	✓	✓	✓	☐	✓	color

Volume Manager

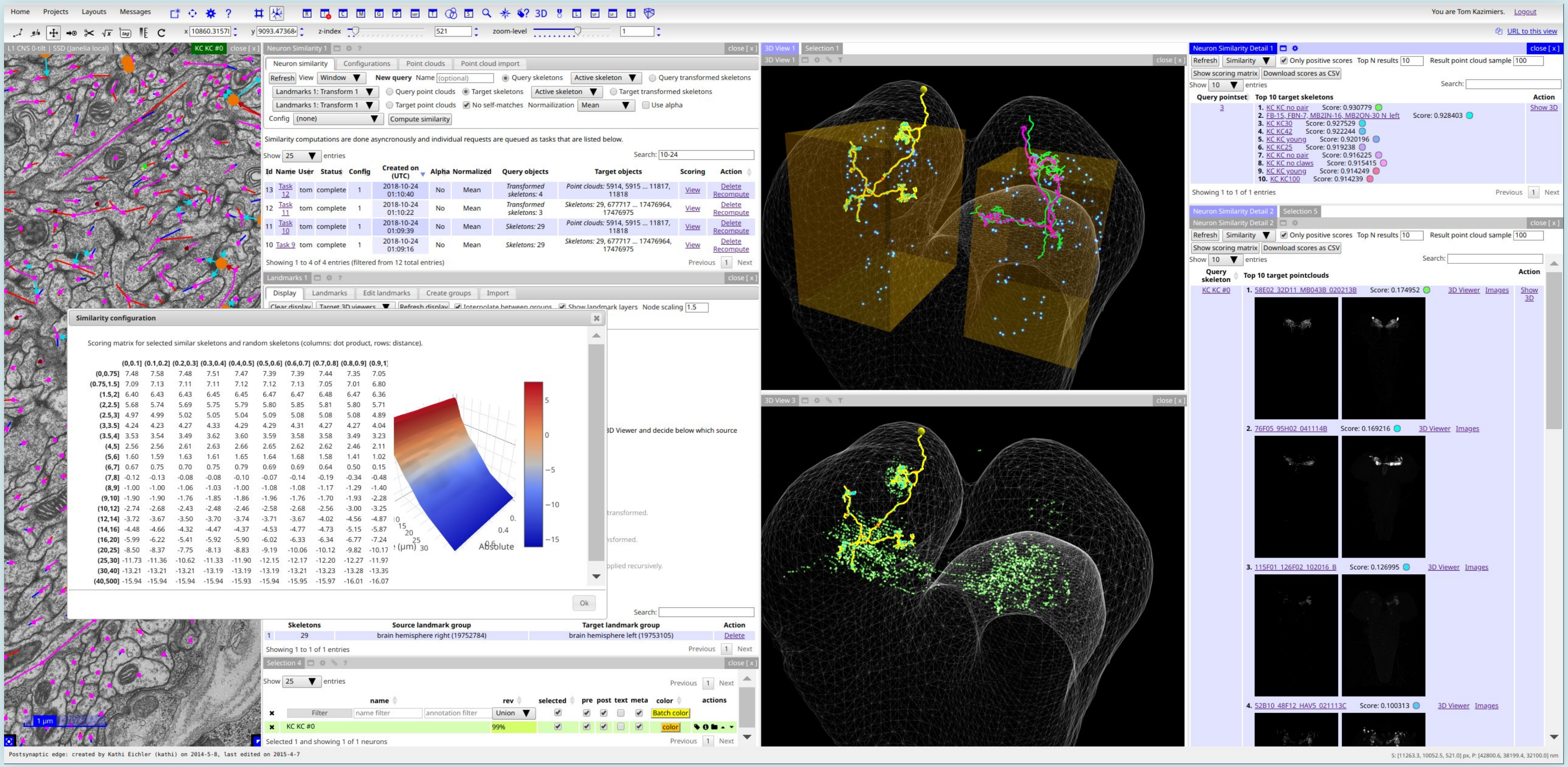
☑	🟥 SLP_R	464	Superior lateral protocerebrum right	Export, Primary	List skeletons List connectors
☐	🟦 SIP_R	465	Superior intermediate protocerebrum right	Export, Primary	List skeletons List connectors
☑	🟪 SMP_R	466	Superior medial protocerebrum right	Export, Primary	List skeletons List connectors

0.000, 0.000 -> 1.000,1.000

S: [94860.8, 41480.5, 4798.0] px, P: [379443.1, 165922.1, 191920.0] nm

Typical CATMAID workspace with 2D and 3D views plus some connectivity tools
Example circuit from Dolan et al. 2019, FAFBv14 dataset

USER INTERFACE II



Similarity computations are done asyncronously and individual requests are queued as tasks that are listed below.

Id	Name	User	Status	Config	Created on (UTC)	Alpha	Normalized	Query objects	Target objects	Scoring	Action
13	Task 12	tom	complete	1	2018-10-24 01:10:40	No	Mean	*Transformed skeletons:* 4	*Point clouds:* 5914, 5915 ... 11817, 11818	View	Delete Recompute
12	Task 11	tom	complete	1	2018-10-24 01:10:22	No	Mean	*Transformed skeletons:* 3	*Skeletons:* 29, 677717 ... 17476964, 17476975	View	Delete Recompute
11	Task 10	tom	complete	1	2018-10-24 01:09:39	No	Mean	*Skeletons:* 29	*Point clouds:* 5914, 5915 ... 11817, 11818	View	Delete Recompute
10	Task 9	tom	complete	1	2018-10-24 01:09:16	No	Mean	*Skeletons:* 29	*Skeletons:* 29, 677717 ... 17476964, 17476975	View	Delete Recompute

Showing 1 to 4 of 4 entries (filtered from 12 total entries)

Similarity configuration

Scoring matrix for selected similar skeletons and random skeletons (columns: dot product, rows: distance).

	(0,0.1]	(0.1,0.2]	(0.2,0.3]	(0.3,0.4]	(0.4,0.5]	(0.5,0.6]	(0.6,0.7]	(0.7,0.8]	(0.8,0.9]	(0.9,1]
(0,0.75]	7.48	7.58	7.48	7.51	7.47	7.39	7.39	7.44	7.35	7.05
(0.75,1.5]	7.09	7.13	7.11	7.11	7.12	7.12	7.13	7.05	7.01	6.80
(1.5,2]	6.40	6.43	6.43	6.45	6.45	6.47	6.47	6.48	6.47	6.36
(2,2.5]	5.68	5.74	5.69	5.75	5.79	5.80	5.85	5.81	5.80	5.71
(2.5,3]	4.97	4.99	5.02	5.05	5.04	5.09	5.08	5.08	5.08	4.89
(3,3.5]	4.24	4.23	4.27	4.33	4.29	4.29	4.31	4.27	4.27	4.04
(3.5,4]	3.53	3.54	3.49	3.62	3.60	3.59	3.58	3.58	3.49	3.23
(4,5]	2.56	2.56	2.61	2.63	2.66	2.65	2.62	2.62	2.46	2.11
(5,6]	1.60	1.59	1.63	1.61	1.65	1.64	1.68	1.58	1.41	1.02
(6,7]	0.67	0.75	0.70	0.75	0.79	0.69	0.69	0.64	0.50	0.15
(7,8]	-0.12	-0.13	-0.08	-0.08	-0.10	-0.07	-0.14	-0.19	-0.34	-0.48
(8,9]	-1.00	-1.00	-1.06	-1.03	-1.00	-1.08	-1.08	-1.17	-1.29	-1.40
(9,10]	-1.90	-1.90	-1.76	-1.85	-1.86	-1.96	-1.76	-1.70	-1.93	-2.28
(10,12]	-2.74	-2.68	-2.43	-2.48	-2.46	-2.58	-2.68	-2.56	-3.00	-3.25
(12,14]	-3.72	-3.67	-3.50	-3.70	-3.74	-3.71	-3.67	-4.02	-4.56	-4.87
(14,16]	-4.48	-4.66	-4.32	-4.47	-4.37	-4.53	-4.77	-4.73	-5.15	-5.87
(16,20]	-5.99	-6.22	-5.41	-5.92	-5.90	-6.02	-6.33	-6.34	-6.77	-7.24
(20,25]	-8.50	-8.37	-7.75	-8.13	-8.83	-9.19	-10.06	-10.12	-9.82	-10.17
(25,30]	-11.73	-11.36	-10.62	-11.33	-11.90	-12.15	-12.17	-12.20	-12.27	-11.97
(30,40]	-13.21	-13.21	-13.21	-13.19	-13.19	-13.19	-13.21	-13.23	-13.28	-13.39
(40,500]	-15.94	-15.94	-15.94	-15.94	-15.93	-15.94	-15.95	-15.97	-16.01	-16.07

	Skeletons	Source landmark group	Target landmark group	Action
1	29	brain hemisphere right (19752784)	brain hemisphere left (19753105)	Delete

Showing 1 to 1 of 1 entries

Neuron Similarity Detail 1

Query pointset	Top 10 target skeletons	Action
3	1. KC KC no pair Score: 0.930779 ● 2. FB-15, FBN-7, MB2IN-16, MB2ON-30 N left Score: 0.928403 ● 3. KC KC30 Score: 0.927529 ● 4. KC KC42 Score: 0.922244 ● 5. KC KC young Score: 0.920196 ● 6. KC KC25 Score: 0.919238 ● 7. KC KC no pair Score: 0.916225 ● 8. KC KC no claws Score: 0.915415 ● 9. KC KC young Score: 0.914249 ● 10. KC KC100 Score: 0.914239 ●	Show 3D

Showing 1 to 1 of 1 entries

Neuron Similarity Detail 2

Query skeleton	Top 10 target pointclouds	Action
KC KC #0	1. 58E02_32D11_MB043B_020213B Score: 0.174952 ● 3D Viewer Images 2. 76F05_95H02_041114B Score: 0.169216 ● 3D Viewer Images 3. 115F01_126F02_102016_B Score: 0.126995 ● 3D Viewer Images 4. 52B10_48F12_HAV5_021113C Score: 0.100313 ● 3D Viewer Images	Show 3D

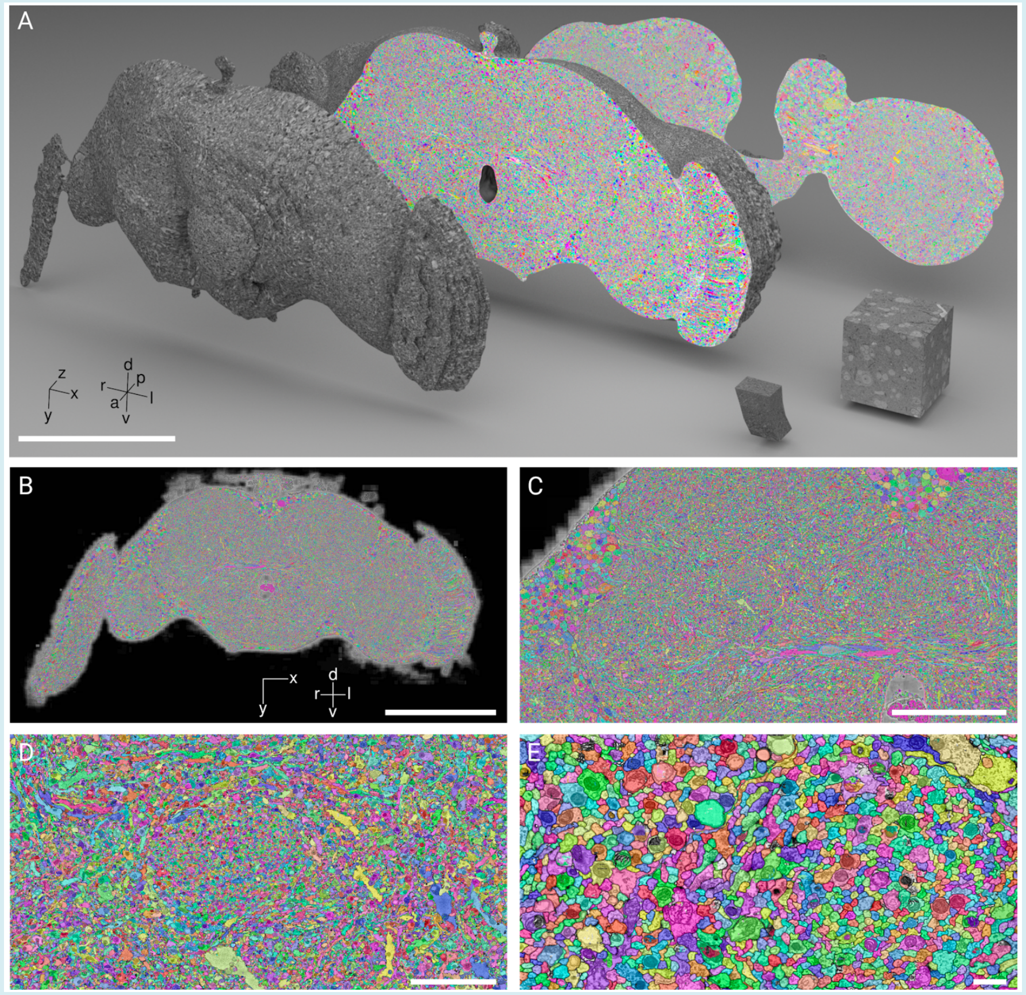
	name	rev	selected	pre	post	text	meta	color	actions
×	KC KC #0	99%	✓	✓	✓		✓	color	

Selected 1 and showing 1 of 1 neurons

```
Postsynaptic edge: created by Kathi Eichler (kathi) on 2014-5-8, last edited on 2015-4-7
```

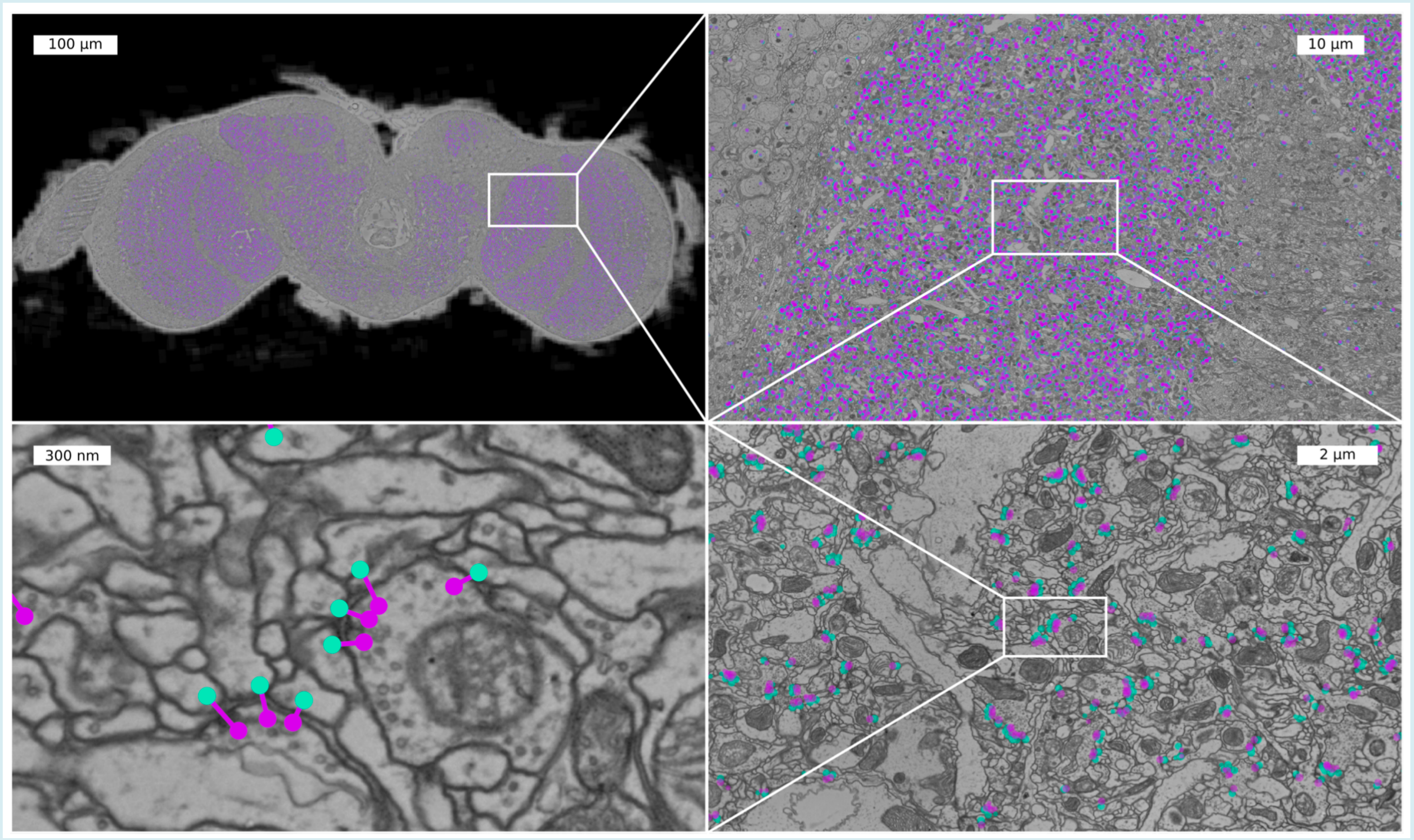
CATMAID with neuron similarity tools and point-clouds from light data
Screenshots from Albert Cardona's Drosophila L1 dataset

NEURON SEGMENTATION



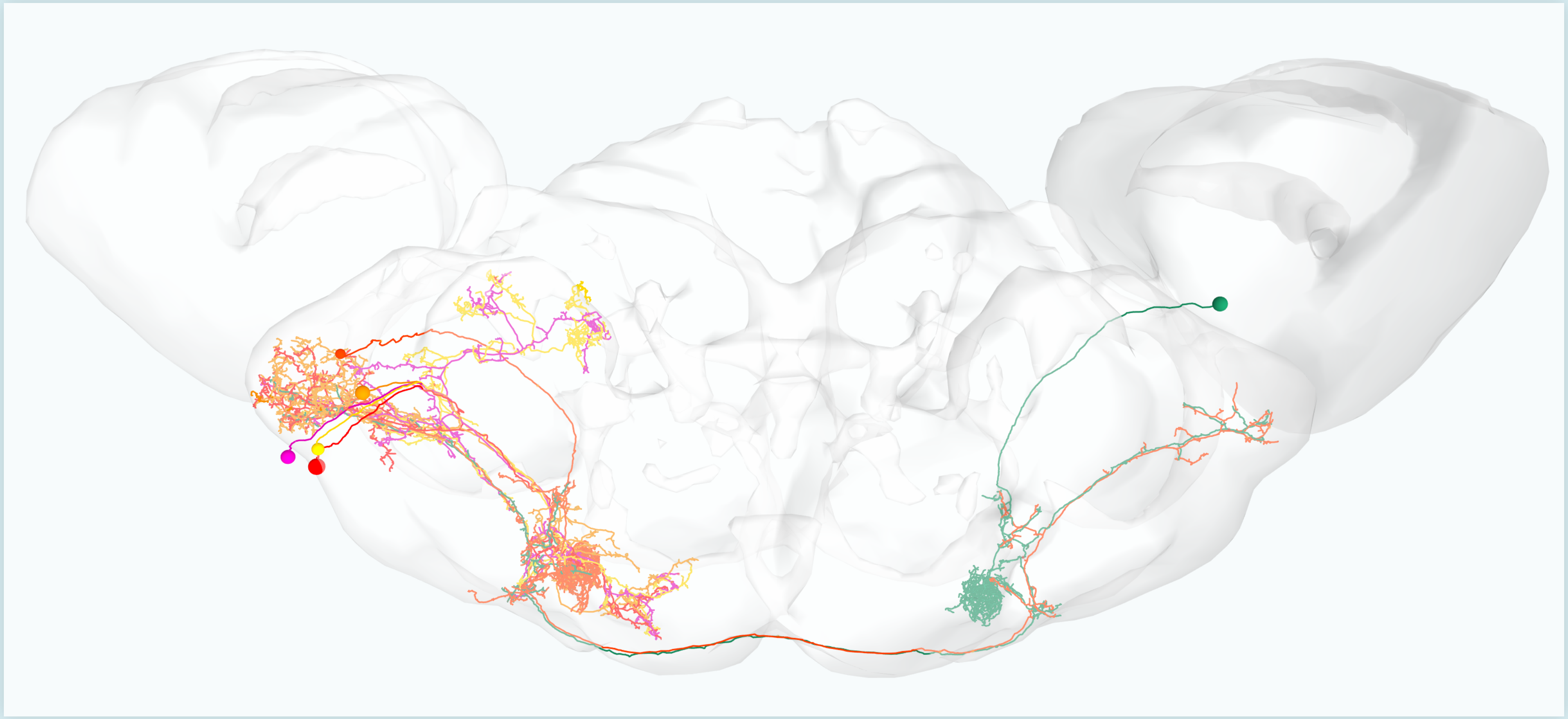
Segmentation of FAFB dataset: used also for skeletonization; Similar in FlyWire project
Li, Jain et al, bioRxiv 2020

SYNAPTIC PARTNER DETECTION



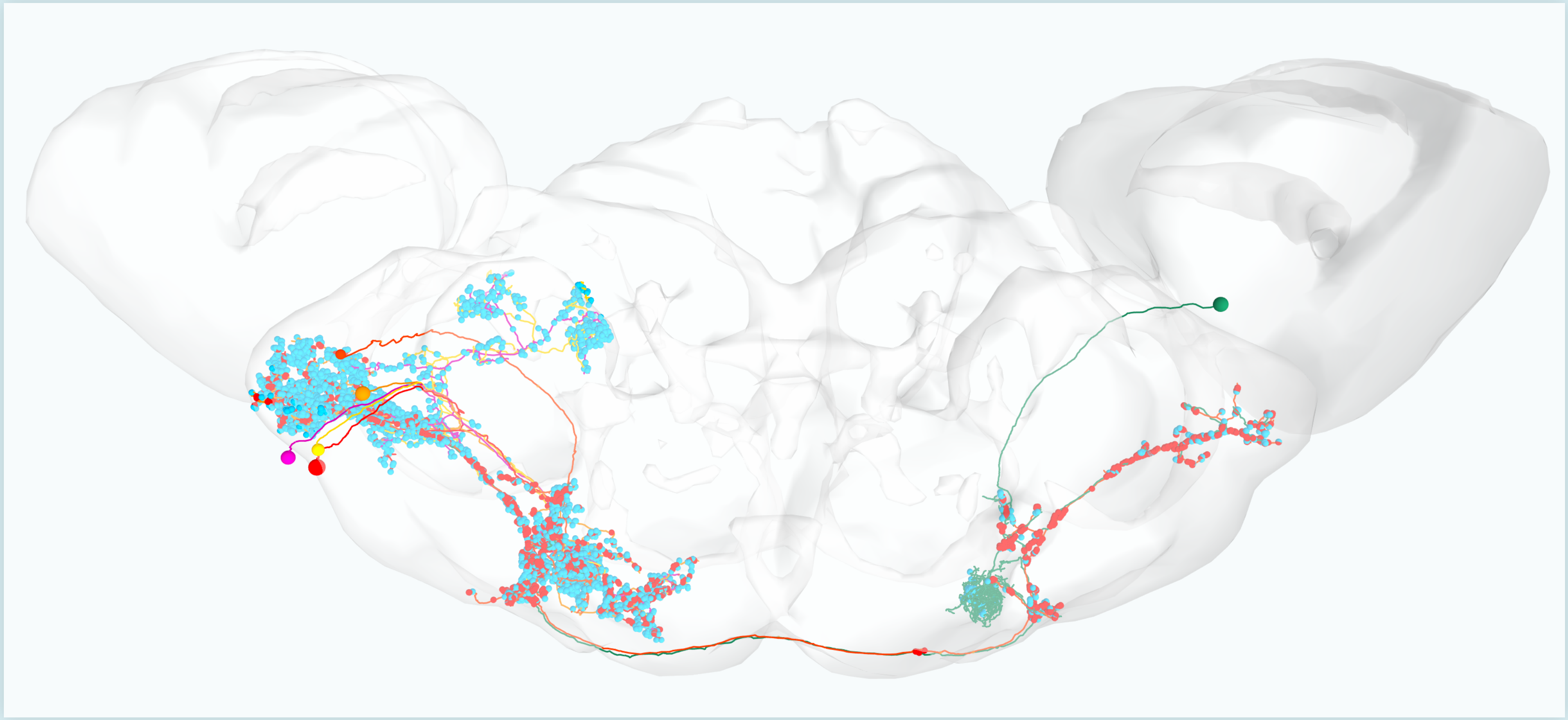
Automatic detection of synaptic partners (CircuitMap tool in CATMAID)
Buhman et al, Nature Methods, 2021

MODELLING NEURONS



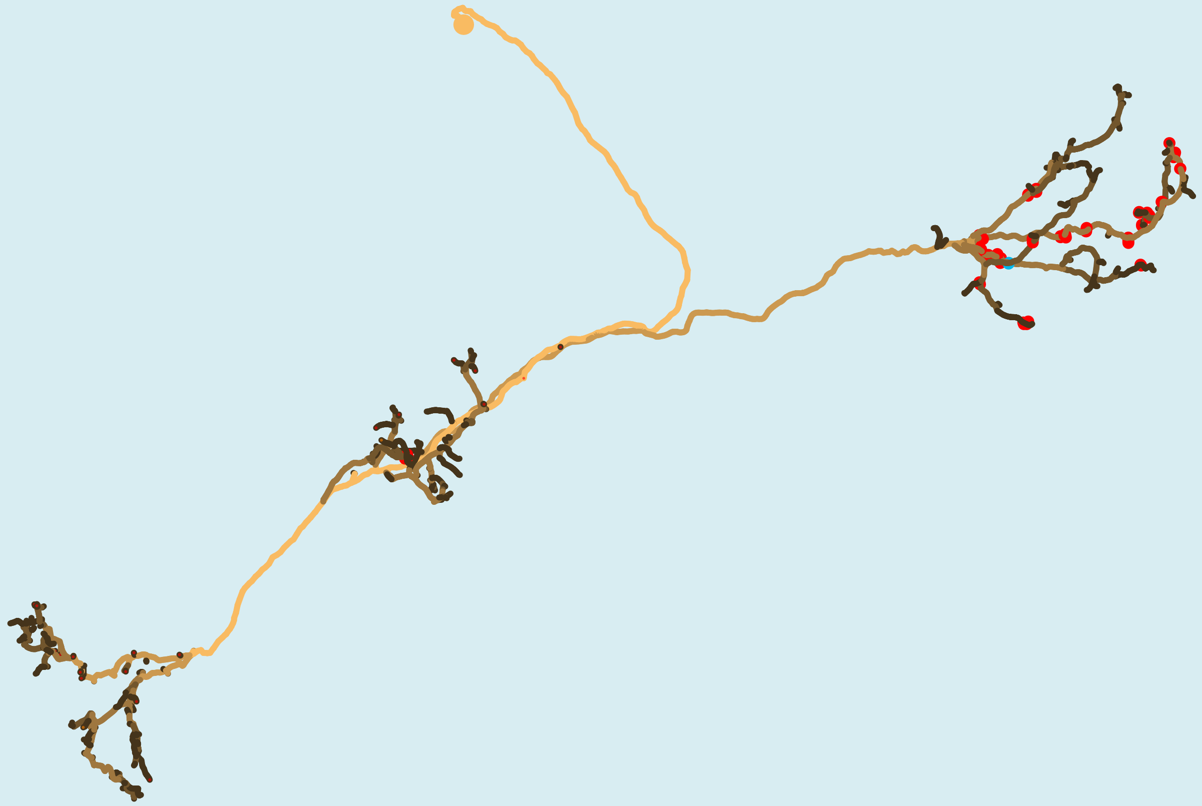
First order (downstream) partners (orange/red) to MBON a2sc Left ASB (green)
From Dolan and Belliart-Guérin et al. 2018

MODELLING NEURONS

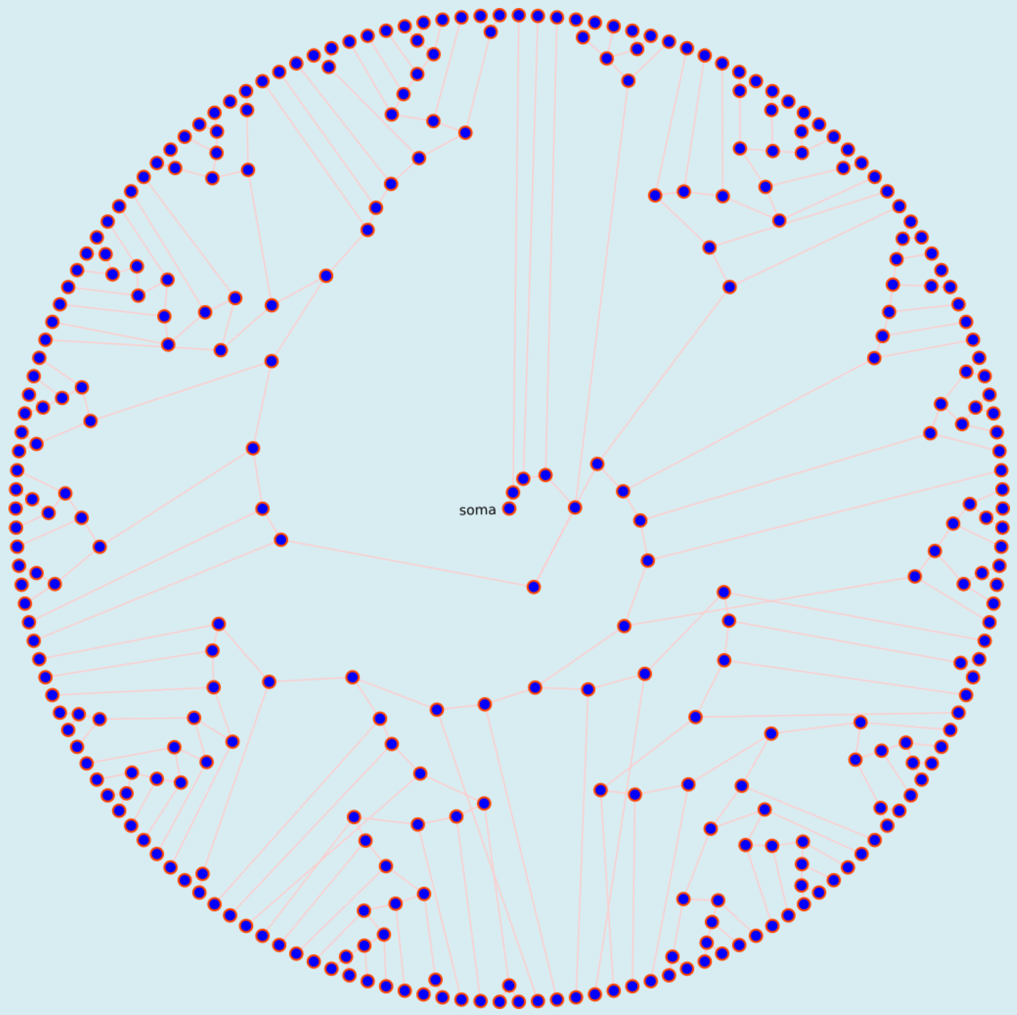


MBON and PD neurons with synapses highlighted (pre: red, post: cyan)
From Dolan and Belliart-Guérin et al. 2018

MORPHOLOGY OF A NEURON



TOPOLOGY OF A NEURON



REPRESENTING NEURONS IN A DATABASE

- Relational data, explicit schema: PostgreSQL
- Nodes of all skeletons in single table
- Every element knows parent or is root (trees)

```
# \d treenode
      Column      |       Type        | Nullable | Default
  ----------------+-------------------+----------+---------
   id             | bigint            | not null |
   parent_id      | bigint            |          |
   skeleton_id    | bigint            | not null |
   project_id     | integer           | not null |
   location_x     | real              | not null |
   …              | …                 | …        |

  Foreign-key constraints:
   "treenode_parent_id_fkey" FOREIGN KEY (parent_id)
                             REFERENCES treenode(id)
```

QUERY NODES OF A SKELETON

- Query ignores any structure/relation between nodes

```
SELECT id, parent_id, location_x, location_y, location_z
FROM treenode
WHERE skeleton_id = {skeleton_id}
```

- Root node of skeleton

```
SELECT id, parent_id, location_x, location_y, location_z
FROM treenode
WHERE skeleton_id = {skeleton_id} AND parent_id IS NULL
```

QUERY NEIGHBOR NODES

```sql
SELECT c.id, c.location_x, c.location_y, c.location_z
FROM treenode c
WHERE parent_id = {node_id}

UNION ALL

SELECT p.id, p.location_x, p.location_y, p.location_z
FROM treenode t
JOIN treenode p
  ON p.id = t.parent_id
WHERE t.id = {node_id}
```

TRAVERSE SKELETONS

- Walk treenode DAG from root to leaves breadth-first, e.g. to find nodes *not* connected to root

```
WITH RECURSIVE nodes (id, depth) AS (
    SELECT t.id, 1
    FROM treenode t
    WHERE t.parent_id IS NULL
      AND t.skeleton_id = {skeleton_id}
  UNION ALL
    SELECT t.id, p.depth + 1
    FROM treenode t
    JOIN nodes p ON t.parent_id = p.id
)
SELECT t.id, t.parent_id
FROM treenode t
WHERE t.skeleton_id = {skeleton_id}
  AND NOT EXISTS (SELECT n.id FROM nodes n WHERE n.id = t.id);
```

TRAVERSE SKELETONS

- Walk treenode DAG from root to leaves breadth-first, e.g. to find nodes *not* connected to root

```
WITH RECURSIVE nodes (id, depth) AS (
    SELECT t.id, 1
    FROM treenode t
    WHERE t.parent_id IS NULL
      AND t.skeleton_id = {skeleton_id}
  UNION ALL
    SELECT t.id, p.depth + 1
    FROM treenode t
    JOIN nodes p ON t.parent_id = p.id
)
SELECT t.id, t.parent_id
FROM treenode t
WHERE t.skeleton_id = {skeleton_id}
  AND NOT EXISTS (SELECT n.id FROM nodes n WHERE n.id = t.id);
```

TRAVERSE SKELETONS

- Walk treenode DAG from root to leaves breadth-first, e.g. to find nodes *not* connected to root

```sql
WITH RECURSIVE nodes (id, depth) AS (
    SELECT t.id, 1
    FROM treenode t
    WHERE t.parent_id IS NULL
      AND t.skeleton_id = {skeleton_id}
  UNION ALL
    SELECT t.id, p.depth + 1
    FROM treenode t
    JOIN nodes p ON t.parent_id = p.id
)
SELECT t.id, t.parent_id
FROM treenode t
WHERE t.skeleton_id = {skeleton_id}
  AND NOT EXISTS (SELECT n.id FROM nodes n WHERE n.id = t.id);
```

TRAVERSE SKELETONS

- Walk treenode DAG from root to leaves breadth-first, e.g. to find nodes *not* connected to root

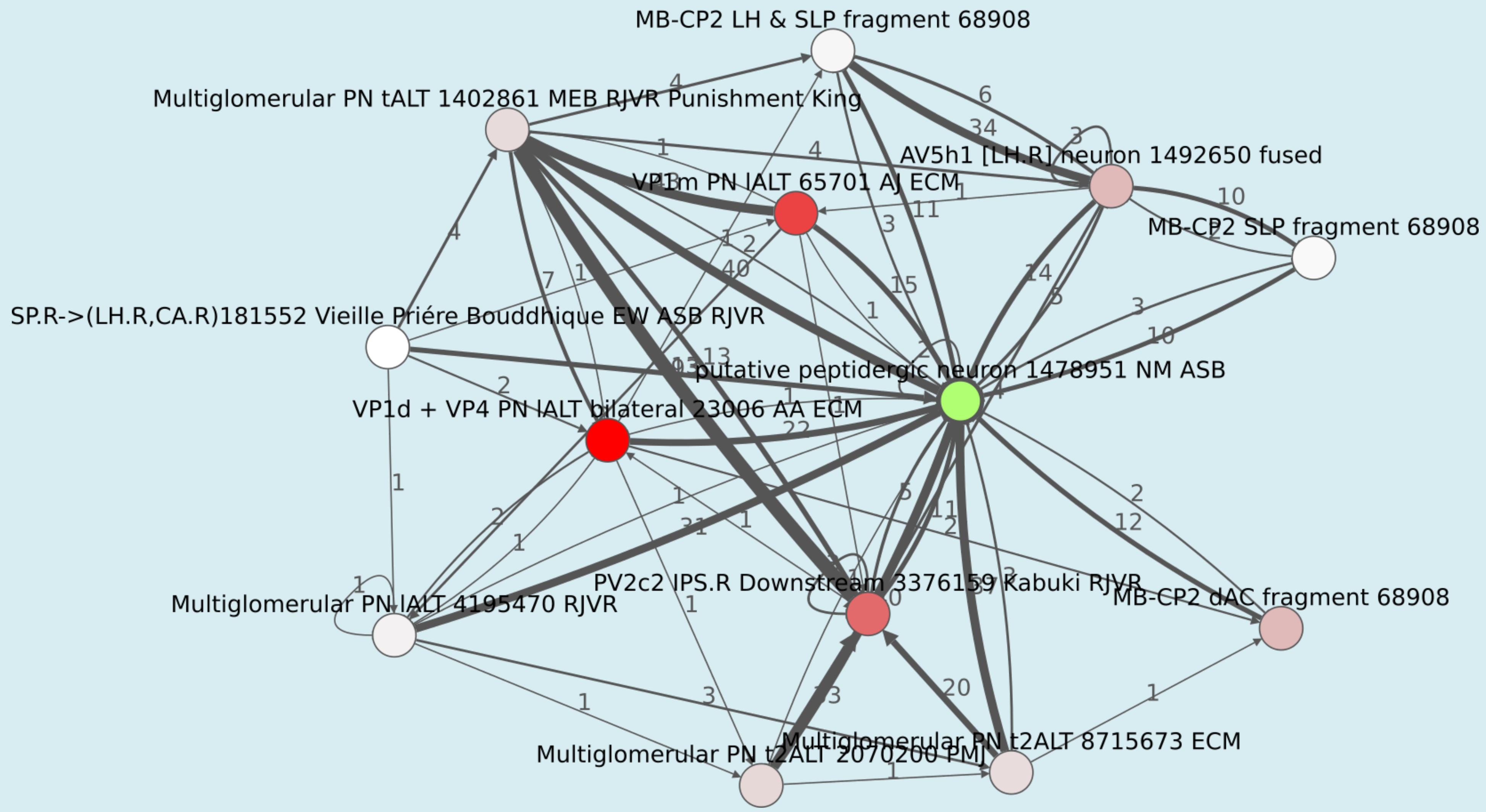
```
WITH RECURSIVE nodes (id, depth) AS (
    SELECT t.id, 1
    FROM treenode t
    WHERE t.parent_id IS NULL
      AND t.skeleton_id = {skeleton_id}
  UNION ALL
    SELECT t.id, p.depth + 1
    FROM treenode t
    JOIN nodes p ON t.parent_id = p.id
)
SELECT t.id, t.parent_id
FROM treenode t
WHERE t.skeleton_id = {skeleton_id}
  AND NOT EXISTS (SELECT n.id FROM nodes n WHERE n.id = t.id);
```

TRAVERSE SKELETONS

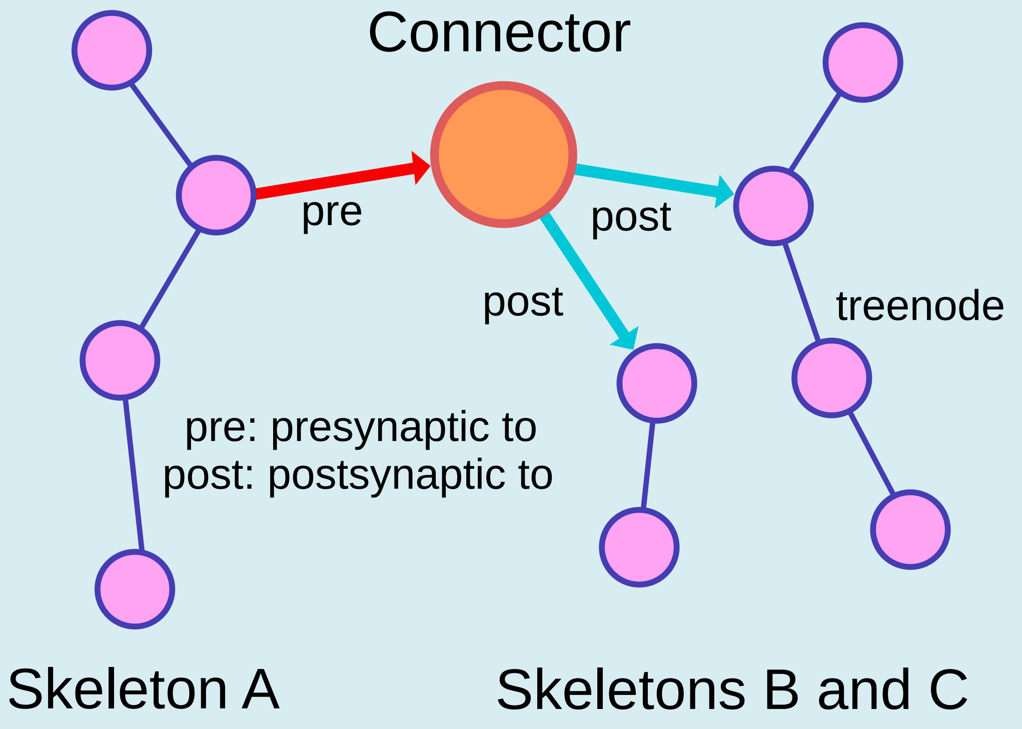
- Walk treenode DAG from root to leaves breadth-first, e.g. to find nodes *not* connected to root

```
WITH RECURSIVE nodes (id, depth) AS (
    SELECT t.id, 1
    FROM treenode t
    WHERE t.parent_id IS NULL
      AND t.skeleton_id = {skeleton_id}
  UNION ALL
    SELECT t.id, p.depth + 1
    FROM treenode t
    JOIN nodes p ON t.parent_id = p.id
)
SELECT t.id, t.parent_id
FROM treenode t
WHERE t.skeleton_id = {skeleton_id}
  AND NOT EXISTS (SELECT n.id FROM nodes n WHERE n.id = t.id);
```

CONNECTED NEURONS


MB-CP2 LH & SLP fragment 68908
Multiglomerular PN tALT 1402861 MEB RJVR Punishment King
AV5h1 [LH.R] neuron 1492650 fused
VP1m PN lALT 65701 AJ ECM
MB-CP2 SLP fragment 68908
SP.R->(LH.R,CA.R)181552 Vieille Priére Bouddhique EW ASB RJVR
putative peptidergic neuron 1478951 NM ASB
VP1d + VP4 PN lALT bilateral 23006 AA ECM
PV2c2 IPS.R Downstream 3376159 Kabuki RJVR
MB-CP2 dAC fragment 68908
Multiglomerular PN lALT 4195470 RJVR
Multiglomerular PN t2ALT 8715673 ECM
Multiglomerular PN t2ALT 2070200 PMJ

NEURON GRAPHS IN CATMAID



NEURON GRAPHS IN SQL

```
# \d treenode_connector
      Column      |       Type        | Nullable | Default
  ---------------+------------------+----------+--------
   relation_id    | bigint            | not null |
   treenode_id    | bigint            | not null |
   connector_id   | bigint            | not null |
   skeleton_id    | bigint            | not null |
   project_id     | integer           | not null |
   confidence     | smallint          | not null | 5
   …              | …                 | …        |

  Foreign-key constraints:
   …
```

FINDING PARTNER NEURONS

- Useful for direct connections between skeletons, e.g. find partner skeleton IDs and synapse counts
- Gets more complex with additional levels

```
SELECT tc2.skeleton_id AS partner, COUNT(*) AS n_post_sites
FROM treenode_conector tc1
JOIN treenode_conector tc2
  ON tc1.connector_id = tc2.connector_id
  AND tc1.treenode_id != tc2.treenode_id
WHERE tc1.relation_id = {presynaptic_to_id}
  AND tc2.relation_id = {postsynaptic_to_id}
  AND tc1.project_id = {project_id}
  AND tc2.project_id = {project_id}
  AND tc1.skeleton_id = {skeleton_id}
GROUP BY tc2.skeleton_id
```

FINDING PARTNER NEURONS

- Useful for direct connections between skeletons, e.g. find partner skeleton IDs and synapse counts
- Gets more complex with additional levels

```
SELECT tc2.skeleton_id AS partner, COUNT(*) AS n_post_sites
FROM treenode_conector tc1
JOIN treenode_conector tc2
  ON tc1.connector_id = tc2.connector_id
  AND tc1.treenode_id != tc2.treenode_id
WHERE tc1.relation_id = {presynaptic_to_id}
  AND tc2.relation_id = {postsynaptic_to_id}
  AND tc1.project_id = {project_id}
  AND tc2.project_id = {project_id}
  AND tc1.skeleton_id = {skeleton_id}
GROUP BY tc2.skeleton_id
```

FINDING PARTNER NEURONS

- Useful for direct connections between skeletons, e.g. find partner skeleton IDs and synapse counts
- Gets more complex with additional levels

```sql
SELECT tc2.skeleton_id AS partner, COUNT(*) AS n_post_sites
FROM treenode_conector tc1
JOIN treenode_conector tc2
  ON tc1.connector_id = tc2.connector_id
  AND tc1.treenode_id != tc2.treenode_id
WHERE tc1.relation_id = {presynaptic_to_id}
  AND tc2.relation_id = {postsynaptic_to_id}
  AND tc1.project_id = {project_id}
  AND tc2.project_id = {project_id}
  AND tc1.skeleton_id = {skeleton_id}
GROUP BY tc2.skeleton_id
```

FINDING PARTNER NEURONS

- Useful for direct connections between skeletons, e.g. find partner skeleton IDs and synapse counts
- Gets more complex with additional levels

```sql
SELECT tc2.skeleton_id AS partner, COUNT(*) AS n_post_sites
FROM treenode_conector tc1
JOIN treenode_conector tc2
  ON tc1.connector_id = tc2.connector_id
  AND tc1.treenode_id != tc2.treenode_id
WHERE tc1.relation_id = {presynaptic_to_id}
  AND tc2.relation_id = {postsynaptic_to_id}
  AND tc1.project_id = {project_id}
  AND tc2.project_id = {project_id}
  AND tc1.skeleton_id = {skeleton_id}
GROUP BY tc2.skeleton_id
```

FINDING PARTNER NEURONS

- Useful for direct connections between skeletons, e.g. find partner skeleton IDs and synapse counts
- Gets more complex with additional levels

```sql
SELECT tc2.skeleton_id AS partner, COUNT(*) AS n_post_sites
FROM treenode_conector tc1
JOIN treenode_conector tc2
  ON tc1.connector_id = tc2.connector_id
  AND tc1.treenode_id != tc2.treenode_id
WHERE tc1.relation_id = {presynaptic_to_id}
  AND tc2.relation_id = {postsynaptic_to_id}
  AND tc1.project_id = {project_id}
  AND tc2.project_id = {project_id}
  AND tc1.skeleton_id = {skeleton_id}
GROUP BY tc2.skeleton_id
```

POSTGIS DATA

- In 2D view: *intersections* of skeletons with 2D plane and store *edges* between nodes
- Main planar access in a single direction (usually Z)

```
# \d treenode_edge

     Column   |          Type           | Collation | Nullable | Default
  ------------+-------------------------+-----------+----------+---------
   id         | bigint                  |           | not null |
   parent_id  | bigint                  |           |          |
   project_id | integer                 |           | not null |
   edge       | geometry(LineStringZ) |           | not null |
  Indexes:
      "treenode_edge_pkey" PRIMARY KEY, btree (id)
      "treenode_edge_project_id_index" btree (project_id)
      "treenode_edge_2d_gist" gist (edge)
      "treenode_edge_3d_gist" gist (edge gist_geometry_ops_nd)
      "treenode_edge_z_range_gist" gist (floatrange(
          st_zmin(edge::box3d), st_zmax(edge::box3d), '[]'::text))
```

BOUNDING BOX QUERIES

- 2D view: cross section of all neurons in a field of view at a particular depth
- Different FoV BBs can benefit from different indexes, so CATMAID tries to use fitting query, e.g.:
  - Whole slice is visible
  - Medium-sized rectangle
  - Small FoV with BB closer to a cube
- For analysis/filters: intersect with 3D meshes (e.g. compartments)

LARGE FIELD OF VIEW

```sql
SELECT *
FROM (
  SELECT DISTINCT ON (id) UNNEST(ARRAY[te.id, te.parent_id]) AS id
  FROM treenode_edge te
  WHERE te.project_id = {project_id}
  AND floatrange(ST_ZMin(te.edge), ST_ZMax(te.edge), '[]')
      && floatrange({z1}, {z2}, '[)')
  AND te.edge && ST_MakeEnvelope({left}, {top}, {right}, {bottom})
  AND ST_3DDWithin(te.edge, ST_MakePolygon(ST_MakeLine(ARRAY[
      ST_MakePoint({left},  {top},    {halfz}),
      ST_MakePoint({right}, {top},    {halfz}),
      ST_MakePoint({right}, {bottom}, {halfz}),
      ST_MakePoint({left},  {bottom}, {halfz}),
      ST_MakePoint({left},  {top},    {halfz})]::geometry[])),
      {halfzdiff})
) bb_treenode
JOIN treenode t1
  ON t1.id = bb_treenode.id
```

Get all child and parent nodes of edges in a 3D bounding box

LARGE FIELD OF VIEW

```sql
SELECT *
FROM (
  SELECT DISTINCT ON (id) UNNEST(ARRAY[te.id, te.parent_id]) AS id
  FROM treenode_edge te
  WHERE te.project_id = {project_id}
  AND floatrange(ST_ZMin(te.edge), ST_ZMax(te.edge), '[]')
      && floatrange({z1}, {z2}, '[)')
  AND te.edge && ST_MakeEnvelope({left}, {top}, {right}, {bottom})
  AND ST_3DDWithin(te.edge, ST_MakePolygon(ST_MakeLine(ARRAY[
      ST_MakePoint({left},  {top},    {halfz}),
      ST_MakePoint({right}, {top},    {halfz}),
      ST_MakePoint({right}, {bottom}, {halfz}),
      ST_MakePoint({left},  {bottom}, {halfz}),
      ST_MakePoint({left},  {top},    {halfz})]::geometry[])),
      {halfzdiff})
) bb_treenode
JOIN treenode t1
  ON t1.id = bb_treenode.id
```

Constrain result edges to those in a user-defined Z range and allow use of index `"treenode_edge_z_range_gist" gist (floatrange( ST_ZMin(edge::box3d), ST_ZMax(edge::box3d), '[]'::text))`

LARGE FIELD OF VIEW

```
SELECT *
FROM (
  SELECT DISTINCT ON (id) UNNEST(ARRAY[te.id, te.parent_id]) AS id
  FROM treenode_edge te
  WHERE te.project_id = {project_id}
  AND floatrange(ST_ZMin(te.edge), ST_ZMax(te.edge), '[]')
      && floatrange({z1}, {z2}, '[)')
  AND te.edge && ST_MakeEnvelope({left}, {top}, {right}, {bottom})
  AND ST_3DDWithin(te.edge, ST_MakePolygon(ST_MakeLine(ARRAY[
      ST_MakePoint({left},  {top},    {halfz}),
      ST_MakePoint({right}, {top},    {halfz}),
      ST_MakePoint({right}, {bottom}, {halfz}),
      ST_MakePoint({left},  {bottom}, {halfz}),
      ST_MakePoint({left},  {top},    {halfz})]::geometry[])),
      {halfzdiff})
) bb_treenode
JOIN treenode t1
  ON t1.id = bb_treenode.id
```

Constrain result edges to only those in XY user-defined XY area and allow use of index `"treenode_edge_2d_gist" gist (edge);`

LARGE FIELD OF VIEW

```
SELECT *
FROM (
  SELECT DISTINCT ON (id) UNNEST(ARRAY[te.id, te.parent_id]) AS id
  FROM treenode_edge te
  WHERE te.project_id = {project_id}
  AND floatrange(ST_ZMin(te.edge), ST_ZMax(te.edge), '[]')
      && floatrange({z1}, {z2}, '[)')
  AND te.edge && ST_MakeEnvelope({left}, {top}, {right}, {bottom})
  AND ST_3DDWithin(te.edge, ST_MakePolygon(ST_MakeLine(ARRAY[
      ST_MakePoint({left},  {top},    {halfz}),
      ST_MakePoint({right}, {top},    {halfz}),
      ST_MakePoint({right}, {bottom}, {halfz}),
      ST_MakePoint({left},  {bottom}, {halfz}),
      ST_MakePoint({left},  {top},    {halfz})]::geometry[])),
      {halfzdiff})
) bb_treenode
JOIN treenode t1
  ON t1.id = bb_treenode.id
```

Test true distance of edge to BB to remove matches where only the BB of the edge is close,
Lowers *false positives* where only query BB and edge BB intersect

SMALL FOV / CUBE-LIKE BB

```
SELECT *
FROM (
  SELECT DISTINCT ON (id) UNNEST(ARRAY[te.id, te.parent_id]) AS id
  FROM treenode_edge te
  WHERE te.project_id = {project_id}
  AND te.edge &&& ST_MakeLine(ARRAY[
      ST_MakePoint({left}, {bottom}, {z2}),
      ST_MakePoint({right}, {top}, {z1})] ::geometry[])
  AND ST_3DDWithin(te.edge, ST_MakePolygon(ST_MakeLine(ARRAY[
      ST_MakePoint({left},  {top},    {halfz}),
      ST_MakePoint({right}, {top},    {halfz}),
      ST_MakePoint({right}, {bottom}, {halfz}),
      ST_MakePoint({left},  {bottom}, {halfz}),
      ST_MakePoint({left},  {top},    {halfz})]::geometry[])),
      {halfzdiff})
) bb_treenode
JOIN treenode t1
  ON t1.id = bb_treenode.
```

Similar structure like other query, but different index
Small result set, no extra filter conditions are beneficial

SMALL FOV / CUBE-LIKE BB

```
SELECT *
FROM (
  SELECT DISTINCT ON (id) UNNEST(ARRAY[te.id, te.parent_id]) AS id
  FROM treenode_edge te
  WHERE te.project_id = {project_id}
  AND te.edge &&& ST_MakeLine(ARRAY[
      ST_MakePoint({left}, {bottom}, {z2}),
      ST_MakePoint({right}, {top}, {z1})] ::geometry[])
  AND ST_3DDWithin(te.edge, ST_MakePolygon(ST_MakeLine(ARRAY[
      ST_MakePoint({left},  {top},    {halfz}),
      ST_MakePoint({right}, {top},    {halfz}),
      ST_MakePoint({right}, {bottom}, {halfz}),
      ST_MakePoint({left},  {bottom}, {halfz}),
      ST_MakePoint({left},  {top},    {halfz})]::geometry[])),
      {halfzdiff})
) bb_treenode
JOIN treenode t1
  ON t1.id = bb_treenode.
```

The &&& operator allows the planner to use the index
`"treenode_edge_3d_gist" gist (edge gist_geometry_ops_nd)`

SMALL FOV / CUBE-LIKE BB

```
SELECT *
FROM (
  SELECT DISTINCT ON (id) UNNEST(ARRAY[te.id, te.parent_id]) AS id
  FROM treenode_edge te
  WHERE te.project_id = {project_id}
  AND te.edge &&& ST_MakeLine(ARRAY[
      ST_MakePoint({left}, {bottom}, {z2}),
      ST_MakePoint({right}, {top}, {z1})] ::geometry[])
  AND ST_3DDWithin(te.edge, ST_MakePolygon(ST_MakeLine(ARRAY[
      ST_MakePoint({left},  {top},    {halfz}),
      ST_MakePoint({right}, {top},    {halfz}),
      ST_MakePoint({right}, {bottom}, {halfz}),
      ST_MakePoint({left},  {bottom}, {halfz}),
      ST_MakePoint({left},  {top},    {halfz})]::geometry[])),
      {halfzdiff})
) bb_treenode
JOIN treenode t1
  ON t1.id = bb_treenode.
```

Limit the number of false positive BB-only intersections, just like before

CLOSEST NODE IN 3D

```sql
SELECT treenode.id, skeleton_id, location_x, location_y, location_z
FROM treenode
JOIN (
    SELECT id, edge
    FROM treenode_edge
    WHERE project_id = {project_id}
    ORDER BY edge <<->> ST_MakePoint({x}, {y}, {z})
    LIMIT 100
) closest_node(id, edge)
ON closest_node.id = treenode.id
ORDER BY ST_StartPoint(edge) <<->> ST_MakePoint({x}, {y}, {z})
LIMIT 1
```

Find closest node among the 100 closest edge centeroids in order to allow 3D index use.
Assume closest node is among closest edge BB centroids (`<<->> operator`)

CLOSEST NODE IN 3D

```
SELECT treenode.id, skeleton_id, location_x, location_y, location_z
FROM treenode
JOIN (
    SELECT id, edge
    FROM treenode_edge
    WHERE project_id = {project_id}
    ORDER BY edge <<->> ST_MakePoint({x}, {y}, {z})
    LIMIT 100
) closest_node(id, edge)
ON closest_node.id = treenode.id
ORDER BY ST_StartPoint(edge) <<->> ST_MakePoint({x}, {y}, {z})
LIMIT 1
```

Find 100 closest (euclidean) edges

CLOSEST NODE IN 3D

```sql
SELECT treenode.id, skeleton_id, location_x, location_y, location_z
FROM treenode
JOIN (
    SELECT id, edge
    FROM treenode_edge
    WHERE project_id = {project_id}
    ORDER BY edge <<->> ST_MakePoint({x}, {y}, {z})
    LIMIT 100
) closest_node(id, edge)
ON closest_node.id = treenode.id
ORDER BY ST_StartPoint(edge) <<->> ST_MakePoint({x}, {y}, {z})
LIMIT 1
```

Get closest edge start node of closest edges

MORE POSTGIS

- More spatial annotations (POIs, bookmarks, text)
- Storing 3D meshes as `TIN` data
- Store spatial meta data for point clouds (BB, center of mass)

Thank you PostGIS team for making spatial data in the DB easier, even beyond GIS!

ACKNOWLEDGMENTS

CONTRIBUTORS

- Albert Cardona
- Andrew Champion
- Chris Barnes
- Stephan Gerhard
- Davi Bock
- Stephan Saalfeld
- Marta Costa
- Greg Jefferis
- Will Patton
- Philipp Schlegel
- Alex Bates

DATA CREATION AND DATA WRANGLING

- Cardona Lab Tracers
- Wellcome Trust Tracers
- CAT Team @ JRC
- Eric Perlman @ Yikes, LLC

LABS AND TEAMS

- Bock Lab @ UVM
- Cardona Lab @ LMB
- Lippincott-Schwartz Lab @ JRC
- Jekely Lab @ UEX
- Lee Lab @ HMS
- Funke Lab @ JRC

QUESTIONS?

- catmaid.org
- github.com/catmaid/catmaid
- tom@kazmos.de
- @tomkazimiers
- Docker image:

```
docker run -p 8080:80 catmaid/catmaid
```