

Manual for PySeg system installation and usage (v 1.0)

Antonio Martinez-Sanchez^{*} and Vladan Lucic

January 9, 2020

^{*} an.martinez.s.sw@gmail.com (martinez@biochem.mpg.de)

Contents

1	Introduction	1
2	System requirements	2
3	Installation	3
3.1	Uninstall	4
4	Testing (tutorial exp_ssmb)	4
4.1	Synthetic grid	4
4.2	Bio-material tracing in cryo-ET tomograms for single membrane analysis	5
4.2.1	Computing the Spatial Embedded Graph (GraphMCF)	5
4.2.2	Graph refinement (filament network)	6
4.2.3	Particle picking	7
4.3	Unsupervised and deterministic classification	8
4.4	Statistical spatial analysis	9
4.4.1	Input preparation	10
4.4.2	Univariate first order analysis	10
4.4.3	Univariate second order analysis	11
4.4.4	Bivariate second order metrics	12
	References	13

1 Introduction

PySeg system (code) contains two Python libraries called **pyseg** and **pyorg** for processing and analyzing data in cryo-Electron Tomography (cryo-ET), besides some additional dependencies not included in standard Linux systems. Library **pyseg** contains functionality for reference free picking proteins in cryo-ET tomograms and **pyorg** for their statistical analysis, in general we will call them

both together as PySeg. Additionally some standalone scripts and input data are provided for testing and illustrative purposes.

IMPORTANT: it is required to be proficient with Python programming language for taking advantage of all capabilities contained by PySeg, unfortunately no GUI is available yet. PySeg is an open source project so any contribution is very welcomed.

2 System requirements

The installers provided in this release has been tested in a Ubuntu 18.04 machine. However, PySeg system can also be installed in any other Linux 64bit platform if all dependencies are satisfied.

System prerequisites:

- Anaconda (tested version 4.3.21), package manager for Python (v 2.7.13)
- CMAKE (tested version 2.812.2)
- Ubuntu packages required by DisPerSe, if you have already a installed version of DisPerSe they may not be needed:
 - Qt5
 - M4
 - Boost
 - GCC/G++
 - MathGL
 - GSL
 - gmp
 - VTK
 - CGAL
- Ubuntu packages required by graph-tool, if you have already a installed version of graph-tool they may not be needed:
 - pycairo

PySeg has some other dependencies not included in Ubuntu, advanced users can install them separately, but they are included in this release with a tested installer to simplify the system installation.

External software included in PySeg system:

- DisPerSe (v 0.9.24) [10]
- Python packages (are downloaded/installed through Anaconda package manager):
 - Numpy
 - Scipy

- Scikit-learn [5]
- scikit-image
- scikit-fmm
- Pyfits
- OpenCV
- Graph-tool [6]
- beautifulsoup4
- lxml
- pillow
- vtk
- Python packages already included:
 - Pyto [1] (some testing data were removed to reduce disk space)

Next software is not required to run PySeg, nevertheless it is strongly recommended to analyze/visualize its results:

- Paraview (v 5.3.0)
- IMOD (v 4.9.0) [4]
- Chimera (v 1.10.2) [7]
- Relion (v 2.1) [9]

3 Installation

Copy or extract the **python_system** folder in the path you want to install the whole system. The installation of all dependencies to run PySeg on a Ubuntu 18.04 computer requires the next steps (other systems may need some modifications for the installers):

1. Make sure that all scripts for installation have execution privileges:
 - `path/to/pyseg_system/sys>chmod u+x *.sh`
2. Install Ubuntu APT packages, it requires administrative privileges:
 - `path/to/pyseg_system/sys>./install_ubuntu_apt_pkgs.sh`
3. Install third party dependencies (currently just to compile locally DisPerSe, not required if you only want to use **pyorg**):
 - `path/to/pyseg_system/sys>./install_third_parties.sh`
4. Install conda, if you have already a conda version installed on your computer this step may not be required. This step actually installs the latest version for Miniconda2, and follow the interactive instruction (choose the default values, and don't forget to close and open the terminal in order to make the changes have effect):

- `path/to/pyseg_system/sys>./install_miniconda.sh`

5. Install conda packages (graph-tool included):

- `path/to/pyseg_system/sys>./install_conda_pkgs.sh`

6. Set environment variables in `.bashrc` file:

- `path/to/pyseg_system/sys>./set_bashrc_env.sh`

A copy of TomoSegMemTV (with SynapSegTools) is included in `pyseg_system/code`, this a Matlab package for membrane segmentation in (Cryo-)Electron Tomography, in the folders `pyseg_system/docs/{tomosegmentv,synap` you have the documentation for their installation and usage.

3.1 Uninstall

For deleting completely the code, data and the installed software just remove the directory `pyseg_system`. If you want to delete only the installed software and keep the code and data, remove the directory `pyseg_system/sys/soft`. Don't forget to erase the `Pyseg system` entry from your `~/.bashrc` file.

Run the script `pyseg_system/clean_out_data.sh` when you want to delete the output files generated by the `pyseg_system/code/test/*.sh` scripts (see Section 4) and release disk space.

4 Testing (tutorial exp_ssmb)

Here is described how to run a serial of scripts for test that Pyseg has been installed successfully, they are within `pyseg_system/code/tests/*.sh`, you may need to give execution permissions first by running `pyseg_system/code/tests>chmod u+x *.sh`. Before running the testing scripts you need to run `pyseg_system/clean_out_data.sh` to ensure that output directories are available under `pyseg_system/data/`, remember that this script also erases any previous output file.

These tests, require some minimal input experimental data, if you want learn more in deep how to use PySeg and take advantage of its capabilities we encourage to complete the tutorial in `pyseg_system/code/tutorials/synth_ssmb`, their documentation is in `pyseg_system/docs/tutorials/synth_ssmb`, this tutorial generates its own synthetic dataset in order to guarantee the reproducibility of its results, so it is strongly recommended to complete this tutorial before you start to process your experimental data.

All results provided in this document have been generated by a laptop computer with an Intel Core i5 processor and 8Gb RAM. Real experimental datasets in [3] and to complete tutorial in `pyseg_system/code/tutorials/synth_ssmb` we used a computer with 36 processors Intel(R) Xeon(R) and 500 GB of RAM.

Some task are computationally costly and some other are not completely parallelized yet so they can take several minutes. Tasks on big experimental datasets can take hours/days.

4.1 Synthetic grid

This scripts checks that pyseg library can be loaded properly by Python interpreter and evaluates bio-material tracing performance. It generates a set of grid tomograms under different noise conditions and evaluate how precisely is delineated by pyseg software.

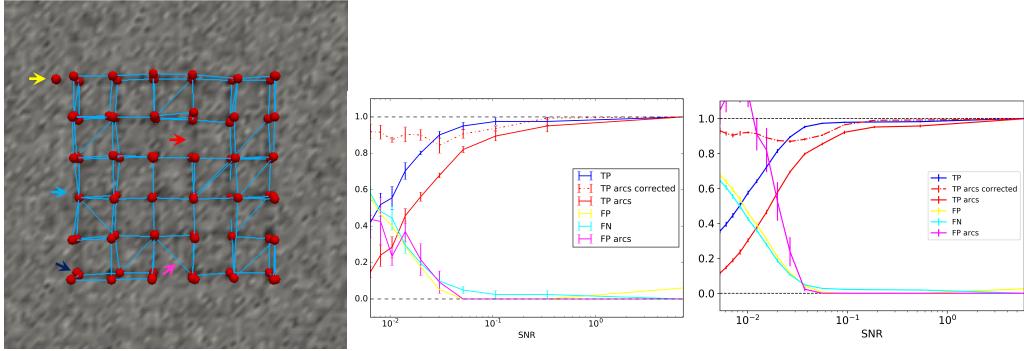


Figure 1: Synthetic grid tomogram (left) with arrows pointing to examples of errors. Test output graphs for short (center) and long (right) versions.

- Location (bash): `pyseg_system/code/tests/tracing_grid.sh`
- Input parameters:
 - `do_long`: short version is 'False' (default) and takes few minutes but is enough for testing functionality, long version 'True' produces stronger statistics but takes a few hours and require high memory resources.
- Output: an image graph with the precision metrics, the synthetic grid tomograms generated and the graphs computed in folder `pyseg_system/code/tests/results`.

4.2 Bio-material tracing in cryo-ET tomograms for single membrane analysis

Here we provide a set of scripts to trace membrane-bound proteins and extract their localization and relative orientation to membrane. As example, an input tomogram with an ER microsome [8] and its membrane segmentation are also provided [2]. We provide here just one tomogram for simplicity, but scripts I/O is managed by STAR files allowing to many tomograms at once.

You can run all the scripts under this Subsection in their proper order just calling the single bash script `pyseg_system/code/tests/tracing.sh`. The log files are stored in `pyseg_system/code/tests/results`.

4.2.1 Computing the Spatial Embedded Graph (GraphMCF)

- Location (Python): `pyseg_system/code/tutorials/exp_ssmb/tracing(mb_graph_{cyto,lumen}.py)`
- Input data: the STAR file `data/tracing(mb_single/in/in_graph.star` with input tomograms and segmentations to process. A segmentation can be a subvolume of a bigger tomogram, `psSegOff{X,Y,Z}` columns are used to identify the top-left-front segmentation corner on tomograms coordinates.
- Input parameters:
 - `res`: input tomograms voxel size (nm/voxel).

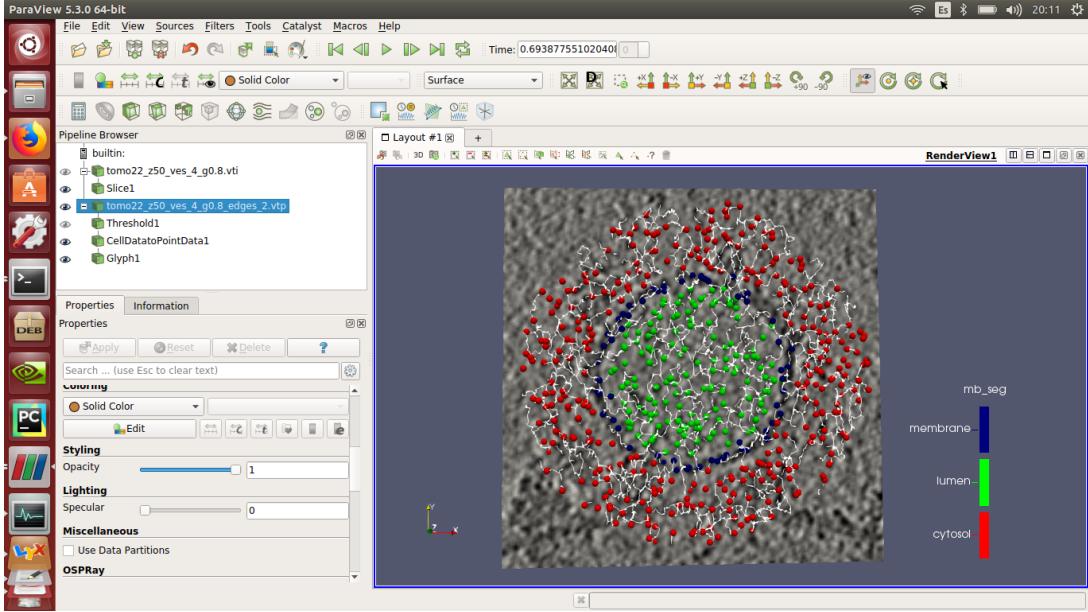


Figure 2: Visualization in Paraview of the spatial embedded graph generated for microsome lumen material.

- **s_sig**: sigma for Gaussian filtering input tomograms, it allows to smooth small and irrelevant features and increases SNR.
- **v_den**: vertex density within membranes, it allows to adjust simplification adaptively for every tomogram.
- **ve_ratio**: averaged ratio vertex/edge in the graph within membrane.
- **max_len**: maximum euclidean shortest distance to membrane in nm.
- Output: a STAR file with the (sub-)tomograms and their corresponding graphs (MbGraphMCF objects) **pyseg_system/data/tutorials/exp_ssmb/tracing(mb_single/graph/{cyto,lumen})/in_graph_mb_graph.star**. Additional files for visualization in Paraview.

IMPORTANT: DisPerSe has a bug and crash with long I/O paths, be careful with that at this point.

4.2.2 Graph refinement (filament network)

In this step the spatially embedded graph can be refined by applying geometrical restrictions. The input graph is reduced to a network of filaments (paths between two vertices), it is required to define the subset of sources and targets vertices then filaments are computed as the closest paths among them. A subset of the graph vertices is defined by a properly constructed XML file (**data/tracing/mb_single/in/mb_sources.xml** and **pyseg_system/data/tutorials/exp_ssmb/tracing/mb_single/in/{cyto,lumen}/{cyto,lumen}**) where **side** indicates the segmented region:

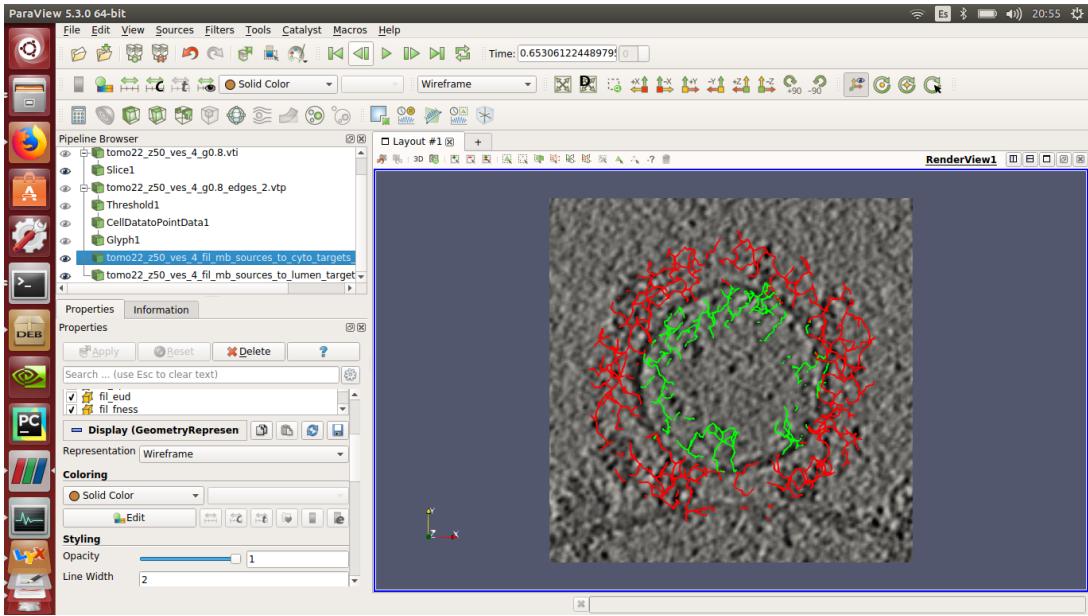


Figure 3: Visualization in Paraview of the cytosolic (red) and luminal (green) filaments.

- Location (Python): `pyseg_system/code/tutorials/exp_ssmb/tracing(mb_files_network_{cyto,lumen}.py)`
- Input data: the STAR file `data/tracing(mb_single/graph/{lumen,cryo}/in_graph_mb_graph.star` as it is returned by `mb_graph.py` script. The XML files for segmenting source and target vertices (see above).
- Filament geometrical parameters:
 - `g_rg_eud`: euclidean distance between source and target vertices.
 - `g_rg_len`: geodesic distance through the graph between source and target vertices.
 - `g_rg_sin`: filament sinuosity, geodesic/euclidean distances ratio.
- Output: A STAR file with filtered version of the input GraphMCF with filaments network `pyseg_system/data/tutorials/exp_ssmb/tracing(mb_single/files/{cyto,lumen}/fil_mb_sources_to_{cyto,lumen}_targets_net.star`. Additional files for visualization in Paraview.

4.2.3 Particle picking

In this step points on input filaments are selected as particle centers for further analysis, mainly for particle reconstruction and feeding a subtomogram averaging procedure. Additionally, information relative to orientation of the particle respect the membrane is also provided to reduce alignment complexity during subtomogram averaging. PySeg has been developed to feed Relion software, an advanced user could use any other software but that would require some modification of the python code.

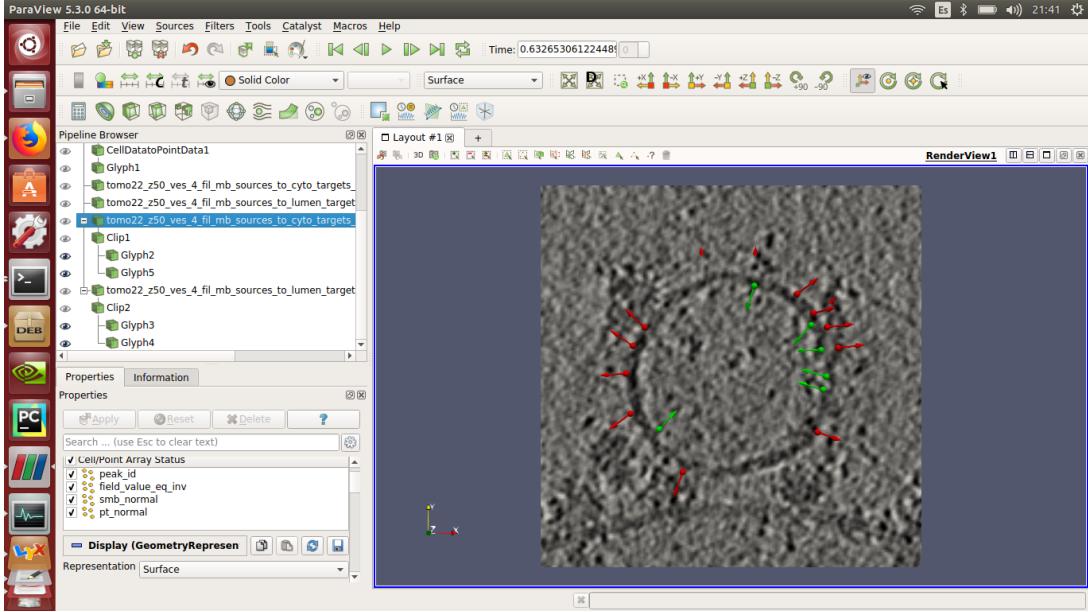


Figure 4: Visualization with Paraview of particles picked in cytosolic (red) and luminal (green) sides, arrows represent the local normal to the segmented membrane.

- Point selection on filaments is defined by a properly constructed XML file
`(pyseg_system/data/tutorials/exp_ssmb/tracing(mb_single/in/{cyto,lumen}/mb_{cyto,lumen}_cont.xml)` where
`cont=+` indicates to pick the point where filament intersect the membrane.
- Location (Python): `pyseg_system/data/tutorials/exp_ssmb/tracing(mb_single(mb_picking_{cyto,lumen}.py`
 - Input data: `pyseg_system/data/tutorials/exp_ssmb/tracing(mb_single/fils/{cyto,lumen}/fil_mb_sources_to_{cyto,lumen}_t`
 the STAR file as it is returned by `mb_fil_network.py` script. A XML file for point selection on
 filaments (see above).
 - Peaks cleaning parameters:
 - `peak_th`: percentile of points to discard by their density level.
 - `peak_ns`: scale suppression in nm, two selected points cannot be closer than this distance.
 - Output: A STAR file with a list with of the coordinates pickled
`pyseg_system/data/tutorials/exp_ssmb/tracing(mb_single/pick/{cyto,lumen}/fil_mb_sources_to_{cyto,lumen}_targets_net`
 additional files for particle reconstruction either in Relion or IMOD with particle centers and
 rotation angles Tilt and Psi (Relion format). Files for visualization in Paraview.

4.3 Unsupervised and deterministic classification

The bash script `classification.sh` calls to a python procedure for structural membrane-bound parti-
 cle classification in an unsupervised, no input for the number of classes and initial reference is

needed, and deterministic manner, the same input the same output. The log files are stored in `pyseg_system/code/tests/results`.

This scripts has two limitations; prior to classification the 3D particles are converted into 2D images by rotational average around membrane normals, and particles must be already aligned respect to the membrane (Tilt and Psi angles in Relion format). To overcome these limitations the procedure has been designed to work in collaboration with Relion, by applying a high symmetrization around membrane normal (C10 for example) we can pre-align the particles with respect the membrane, without considering Rot angle. On the other hand, the script is not designed to produce final 3D averages, its role is to group a highly heterogeneous set of particles, template free picked, which can not be treated directly by standard cryo-ET classification methods, into more homogeneous groups which can be classified and refined by Relion.

IMPORTANT: due to space limitations no experimental data are provided, [3] is an example of the results obtained with this procedure an how to combine it with Relion.

- Location (Python): `pyseg_system/code/tutorials/exp_ssmb/class/plane_align_class_test.py`
- Input data: A STAR file with membrane-bound particles aligned respect the membrane(Tilt and Psi angles)
- Parameters:
 - pp_mask: binary mask to focus the classification.
 - pp_low_sg: sigma (in voxels) for pre-filter particles with a Gaussian kernel, it increases the SNR for each particle and suppress high frequencies.
 - pp_npr: Number of parallel processes.
 - ap_pref: affinity propagation preference (see [5]).
- Output: A set of STAR files with the new classes. Particle exemplar and average per each class.

4.4 Statistical spatial analysis

Here some scripts to quantitatively describe proteins (3D solids) are organized within cellular compartments (irregular volumes) are provided. The basic functionality is implemented in library `pyorg`, and a more detailed description about the metrics computed is in [3].

You can run all the scripts under this Subsection in their proper order just calling the single bash script `pyseg_system/code/tests/org.sh`. The log files are stored in `pyseg_system/code/tests/results`.

For simplicity we are going to take the particles pickled in Section 4.2 as input, nevertheless the ideal situation is to make this analysis after particles has been averaged and classified, then we can analyze separately each protein and use their correct shape.

IMPORTANT: this analysis is completely independent from the picking method, so it is NOT limited to membrane-bound proteins.

IMPORTANT: examples showed here have the intention of introducing the reader in the usage of the software, no biological conclusion can be extracted form them, an example of results obtained from representative experimental dataset can be found in [3].

4.4.1 Input preparation

Particles from many tomograms (ListTomoParticle object) and from different lists, representing different proteins, (SetListTomoParticle object) are gather here into the same object for further analysis.

- Location (Python): `pyseg_system/code/tutorials/exp_ssmb/org/ltomos_generator.py`
- Input data: `pyseg_system/data/tutorials/exp_ssmb/stat/in/in_particles.star` the STAR file with the list of particles which represent different proteins (here cytosolic and luminal particles) and their shape (for simplicity here we use a sphere of radius 2.5), `pyseg_system/data/tutorials/exp_ssmb/stat/in/in_seg.star` the STAR file which pairs segmentations (define the cellular compartments, here the membrane) and tomograms.
- Parameters for pre-processing segmentations:
 - `sg_lbl`: label of the segmented compartment (Volume Of Interest, VOI).
 - `sg_bc`: if True insert only the particles embedded in the VOI.
 - `sg_pj`: particles are projected to VOI surface, recommended for membranes.
 - `sg_origins`: picking and averaging resolutions did not use to agree, this array (an entry for every list) is the scale factor from picking to averaging resolution. They must be introduced to consider properly the alignment shifting computed during sub-tomogram averaging.
 - `sg_dmap_path`: directory for temporary VOI memory maps, if None the VOIs are loaded into memory and it could be overloaded.
- Parameters for post-processing:
 - `pt_res`: voxel size (nm/voxel) used for picking.
 - `pt_ssup`: scale suppression, two particles cannot be closer than this value (particle overlap is not checked here).
 - `pt_min_parts`: discard tomograms with less particles than this value
- Output: A STAR file with the pickled ListTomoParticles objects
`pyseg_system/data/tutorials/exp_ssmb/stat/ltomos/test/test_ssup_8_ltomos.star`. Additional files for visualization are also stored.

4.4.2 Univariate first order analysis

The first order univariate metrics computed are: closest distance histogram and its cumulative distribution function (nearest neighbor distribution), contact function distribution and J-function. Results are computed for every tomogram, then they are averaged for every list of proteins.

- Location (Python): `pyseg_system/code/tutorials/exp_ssmb/org/uni_1st_analysis.py`
- Input data: `pyseg_system/data/tutorials/exp_ssmb/stat/ltomos/test/test_ssup_8_ltomos.star` the STAR file generated by `ltomos_generator.py`.

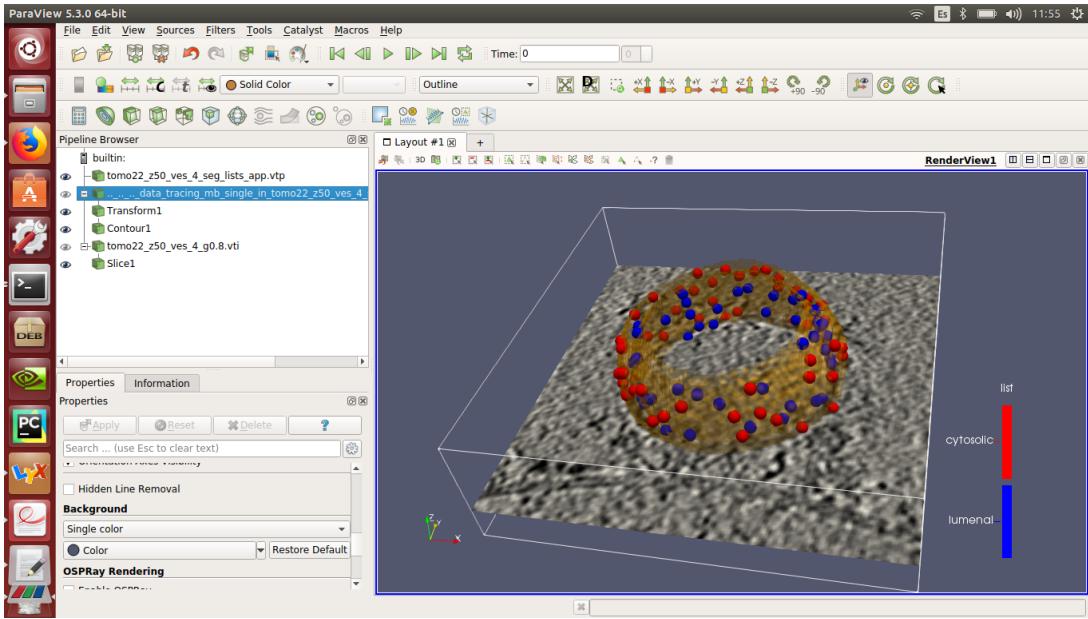


Figure 5: Visualization with Paraview of the VOI (membrane in orange) and two list of particles generated, cytosolic (red) and luminal (green), in both cases protein shape is a 2.5 radius sphere.

- Parameters:
 - **ana_res**: data resolution (nm/voxel).
 - **ana_nbins**: number of bins for histograms.
 - **ana_max**: maximum distance to analyze.
 - **ana_f_npoints**: number of random points to generate for computing contact function distribution.
 - **p_nsims**: number of random simulations per list and tomograms.
 - **p_per**: side percentile out of the displayed confidence intervals.
- Output: A set of graphs with metrics computed by lists and tomograms.

4.4.3 Univariate second order analysis

The second order univariate metrics computed are: Ripley's L and O. Results are computed for every tomogram, then they are averaged for every list of proteins.

- Location (Python): `pyseg_system/code/tutorials/exp_ssmb/org/uni_2nd_analysis.py`
- Input data: `pyseg_system/data/tutorials/exp_ssmb/stat/test/test_sspp_8_itomos.star` the STAR file generated by `Itomos_generator.py`.

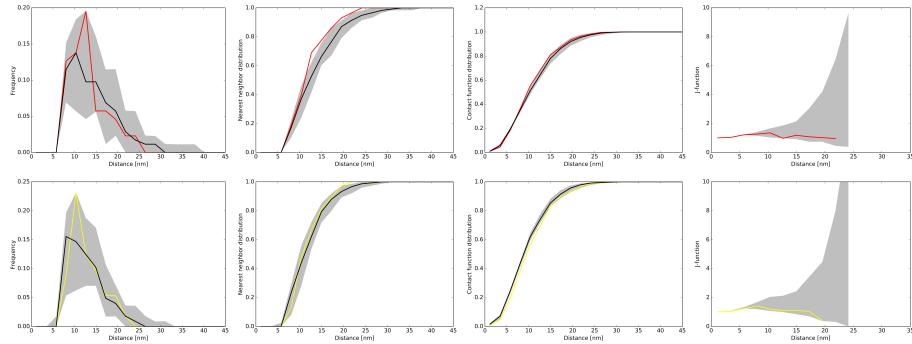


Figure 6: Univariate first order graphs computed. From left to right: closest neighbor histogram, distribution, contact point distribution and J-function. Top row for luminal particles and bottom for cytosolic.

- Parameters:
 - `ana_res`: data resolution (nm/voxel).
 - `ana_rg`: range of distances (in nm) to analyze.
 - `ana_shell_thick`: neighborhood thickness for Ripley's O shell, if None then Ripley's L metric is computed.
 - `ana_conv_iter` and `ana_max_iter`: number of iterations to converge and maximum to try for computing irregular volumes (Monte Carlo based approach). For membranes we recommend `ana_conv_iter=100` and `ana_max_iter=100000`, for cytosolic VOIs these values can be relaxed.
 - `ana_npr`: number of parallel processes to compute Ripley's metrics.
 - `ana_npr_model`: number of parallel processes to generate the random simulations.
 - `p_sims`: number of random simulations per list and tomograms.
 - `p_per`: side percentile out of the displayed confidence intervals.
- Output: A set of graphs with the metrics computed by lists and tomograms.

4.4.4 Bivariate second order metrics

The second order bivariate metrics computed are: Ripley's L and O. Results are computed for every tomogram, then they are averaged for every list of proteins.

- Location (Python): `pyseg_system/code/tutorials/exp_ssmb/org/bi_2nd_analysis.py`
- Input data: `pyseg_system/data/tutorials/exp_ssmb/stat/test/test_sspp_8_itomos.star` the STAR file generated by `Itomos_generator.py`.
- Parameters:

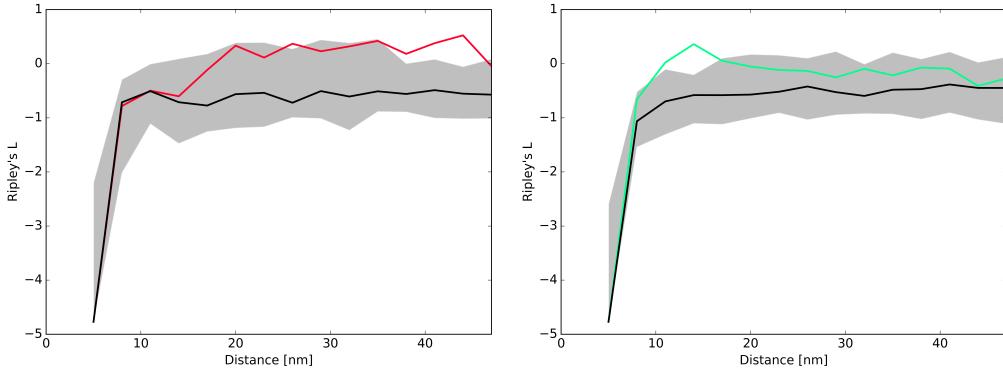


Figure 7: Univariate second order (Ripley's L) graphs computed, luminal left and cytosolic right. Confidence interval [5, 95)% for simulated random model in gray.

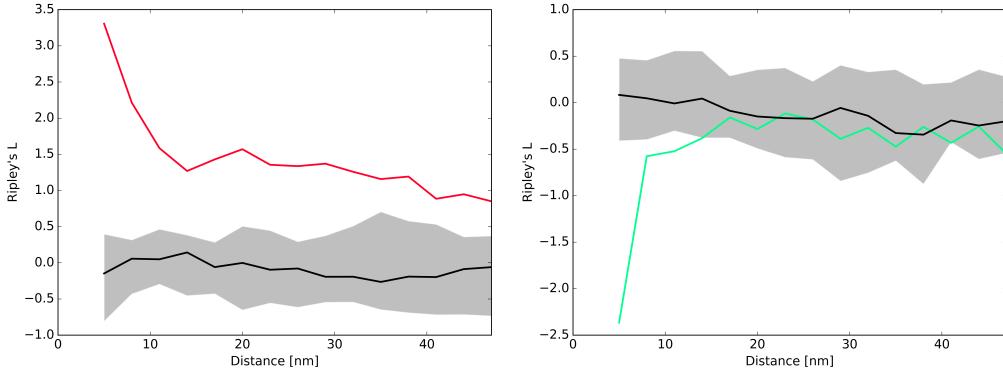


Figure 8: Bivariate second order (Ripley's L) graphs computed, luminal left and cytosolic right. Confidence interval [5, 95)% for simulated random model in gray.

- **in_ref_short_key:** sort key to identify the list used as reference pattern in the bivariate analysis.
- The same as the univariate case (see above).
- Output: A set of graphs with the metrics computed by lists and tomograms.

References

- [1] Vladan Lučić, Rubén Fernández-Busnadio, Ulrike Laugks, and Wolfgang Baumeister. Hierarchical detection and analysis of macromolecular complexes in cryo-electron tomograms using pyto software. *Journal of structural biology*, 196(3):503–514, 2016.

- [2] Antonio Martinez-Sanchez, Inmaculada Garcia, Shoh Asano, Vladan Lucic, and Jose-Jesus Fernandez. Robust membrane detection based on tensor voting for electron tomography. *Journal of structural biology*, 186(1):49–61, 2014.
- [3] Antonio Martinez-Sanchez, Zdravko Kochovski, Ulrike Laugks, Johannes Meyer zum Alten Borgloh, Saikat Chakraborty, Stefan Pfeffer, Wolfgang Baumeister, and Vladan Lučić. Template-free detection and classification of heterogeneous membrane-bound complexes in cryo-electron tomograms. *Nature Methods*, 2020.
- [4] David N Mastronarde and Susannah R Held. Automated tilt series alignment and tomographic reconstruction in imod. *Journal of structural biology*, 197(2):102–113, 2017.
- [5] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- [6] Tiago P Peixoto. The graph-tool python library. *figshare*, 2014.
- [7] Eric F Pettersen, Thomas D Goddard, Conrad C Huang, Gregory S Couch, Daniel M Greenblatt, Elaine C Meng, and Thomas E Ferrin. Ucsf chimera: a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, 2004.
- [8] Stefan Pfeffer, Johanna Dudek, Marko Gogala, Stefan Schorr, Johannes Linxweiler, Sven Lang, Thomas Becker, Roland Beckmann, Richard Zimmermann, and Friedrich Förster. Structure of the mammalian oligosaccharyl-transferase complex in the native er protein translocon. *Nature communications*, 5:3072, 2014.
- [9] Sjors HW Scheres. Relion: implementation of a bayesian approach to cryo-em structure determination. *Journal of structural biology*, 180(3):519–530, 2012.
- [10] Thierry Soubie. The persistent cosmic web and its filamentary structure—i. theory and implementation. *Monthly Notices of the Royal Astronomical Society*, 414(1):350–383, 2011.