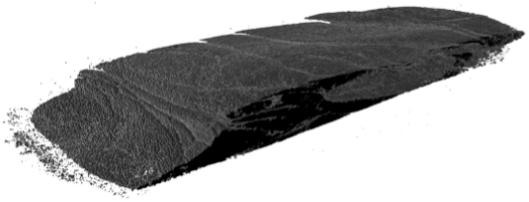
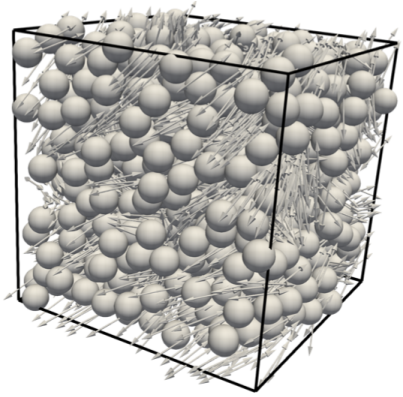
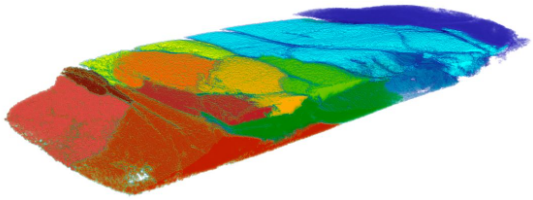
SEGVIZ TOOL USER GUIDE

Pauline Chassonnery*, Diane Peurichard, Sinan Haliyo
Restore Institute, Toulouse III University, France

November 12, 2025

Data visualization using Paraview	
raw *in-vivo* data 	raw *in-silico* data 
segmented *in-vivo* data 	segmented *in-silico* data
Data segmentation using Python	

*E-mail: pauline.chassonnery@ens-cachan.fr

Contents

1 About SegViz

The objective of the SegViz tool is to provide non-specialists in image processing with an integrated, user-friendly tool for the 3D visualisation of systems composed of a mixture of spherical (e.g. cells, particles) and rod-like (e.g. fibers, bacteria) elements and for the automatic detection of spatially connected clusters of spherical objects separated by rod-like elements in such systems. It is aimed in particular at modellers, who may generate such an *in silico* dataset through numerical simulations, and at biologists who may retrieve it from *in vivo* or *in vitro* experiments (for example through 3D microscopy imaging of tissue samples).

Despite recent improvements in the field of high resolution tri-dimensional imaging techniques, the effective 3D visualization of the large datasets remains a challenge. This is particularly the case if a common plateform is to be used for both biological images and mathematical data. Robust scientific visualization tools like the Visualization Toolkit (VTK) and its front-end application, Paraview [4], can bridge the gap between these two types of data. Yet, few efforts have been put in the development of plugins adapted to biological data and models.

On the other hand, objects segmentation and clustering is a wide-spread problem in the domain of image analysis and many methods have been developed to solve it, of which one of the most widely used is the watershed segmentation. But these are generally not well-known to researchers outside this field and the procedures provided by image analysis software usually require a number of preprocessing steps. Moreover, data produced by a mathematical simulation will usually not even be in the form of images and will thus require an extra processing step that can be quite time-consuming if not optimized.

The visualization part of the SegViz tool consists of two Paraview macros, **Sphereviz** and **Rodviz**, described in chapter 3. They take as input a .csv file describing the position, size and other optional properties of a set of spherical (resp. rod-like) objects

and display them as 3D glyph that can be colored according to any property referenced in the dataset. By default, the spherical objects are colored according to their "cluster index" (see below) if that information is available and white otherwise.

Considering that Paraview's preset discrete color maps are either limited to 12 colors or contain colors that are not easy to distinguish, we provide a custom categorical color map based on the work of Sasha Trubetskoy [5]. It contains 20 colors listed in descending order of compatibility with color blindness. This color map can be loaded in any Paraview setup and will be used by the macro **Sphereviz** if it is present. If not, the default Paraview color map "KAAMS" will be used.

The segmenting part of the SegViz tool consists in a Python function, called **WatershedSegmentation**, which is described in chapter 4. It takes as input either a binary image (in the form of a numpy.ndarray or .tiff file) or a list of spherical objects with their properties (in the form of a pandas.DataFrame or .csv file), as well as a number of fine-tuning parameters. In the first case, it returns a labeled image where each region has been attributed a unique label. In the second case, it groups the objects into clusters based on spatial proximity and returns a copy of the pandas.DataFrame with an additional column containing the index of the cluster each object pertains to. In both cases, the segmentation step uses the watershed transformation [6].

Because conventional clustering software are made for biological or physical images, they never include tools to treat the case of periodic boundary conditions. However, this is a very common hypothesis in computational models that needs to be taken into account for clustering. The **WatershedSegmentation** function thus includes an option for periodic boundary conditions.

The SegTools module is free of use, upon proper citation of [2] and of this user manual. The zipped package containing all the necessary files is downloadable here.

2 Installation

2.1 Prerequisites

The SegViz tool requires Python 3.8 or higher, the Python packages numpy, pandas, Pillow and scikit-image, and Paraview 5.4 or higher.

Considering that Python is preinstalled on most computers, first check your eventual version of Python by typing the instruction `python3 --version` in the command prompt window. If this return nothing or a number below 3.8, download the latest Python version from the official website. You can check that the installation went fine by running the instruction `python3 --version` again. See the official download tutorial for more details.

Once this is done, install the required Python packages by typing the instruction `pip install <package-name>` in the command prompt. For instance : `pip install numpy`

The latest release of Paraview can be downloaded from the official website. The minimal requirement for the SegViz tool is Paraview 5.4, which itself requires Python 3.8 to be installed first.

Summary list

- Python >= 3.8
- Python packages : numpy, pandas, Pillow, scikit-image
- Paraview >= 5.4

D'après les instructions de JOSS, le mieux serait de faire un mini-package Python, installable via pip, avec toutes les dépendances prises en charge automatiquement.

2.2 Loading a macro in Paraview

Paraview macros are stored individually in .py files.
To load a macro in your Paraview setup, open the application and go to the menu bar. There, click on the "Import new macro..." entry in the "Macros" drop-down menu (see Figure 1) and select the appropriate .py file in the folder browser. The macro will then be accessible in the "Macros" drop-down menu under the same name as its original file, plus a number if disambiguation is needed.
Note that the macro's file will be copied to Paraview's configuration folder, so that the macro will still be usable if you move or even delete the original file. You can remove a macro from your Paraview configuration using the "Delete..." entry (or, in newer versions of Paraview, the "Edit Macros" entry) in the "Macros" drop-down menu (see Figure 1).

The SegViz tool comprises two macros which can be installed independently, in any order. They are stored in the files `Spherevizz.py` and `Rodviz.py`.

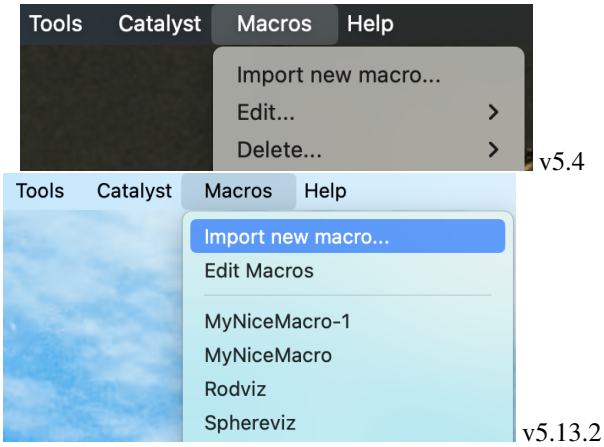


Figure 1: Screenshot of the "Macros" drop-down menu of Paraview menu bar. (redo bottom image with v5.4)

2.3 Loading a color map in Paraview (advanced)

Paraview allows users to import custom color maps from json, xml or ct files. The SegViz tool comes with one such custom color map, based on the work of Sasha Trubetskoy [5] and stored in the file `SashaTrubetskoy.json`. It is used by the macro **Sphereviz** to make it easier to discriminate between different groups of objects in a dataset.
Note that loading this color map isn't necessary for the macro to work (it will default to one of Paraview preset color map), so this somewhat tedious step of the installation can be safely ignored.

Loading a color map requires your Paraview window to be currently using one of its pre-installed color map to render a visual. Follows the instructions in section 3.2 below to display the dataset stored in file `demo_data/dataset_spheres_seg_periodic.csv` using the macro **Sphereviz**.

Once the rendering is done, open the Color Map Editor using the "View > Color Map Editor" entry in the menu bar. Depending on your settings, the editor may open in a side panel or as an additional window.
From the Color Map Editor, open the Presets dialog window by clicking on the "Choose Preset" button (see Figure 2), then click on the "Import" button (see Figure 3) and select the desired file using the folder browser.

Depending on your settings, the new map may not show up immediately, because the current color maps in view are only the default ones. If this is the case, you can change the parameter "Default" to "All" in the Presets window to reveal the new map : it will appear towards the end of the list, with its name in italics. You can select it and tick the "Show this map in Default" box to have it shown by default.
You can test the new color map by manually applying it to your current visual. Afterward, the macro **Sphereviz** will automatically use this map.

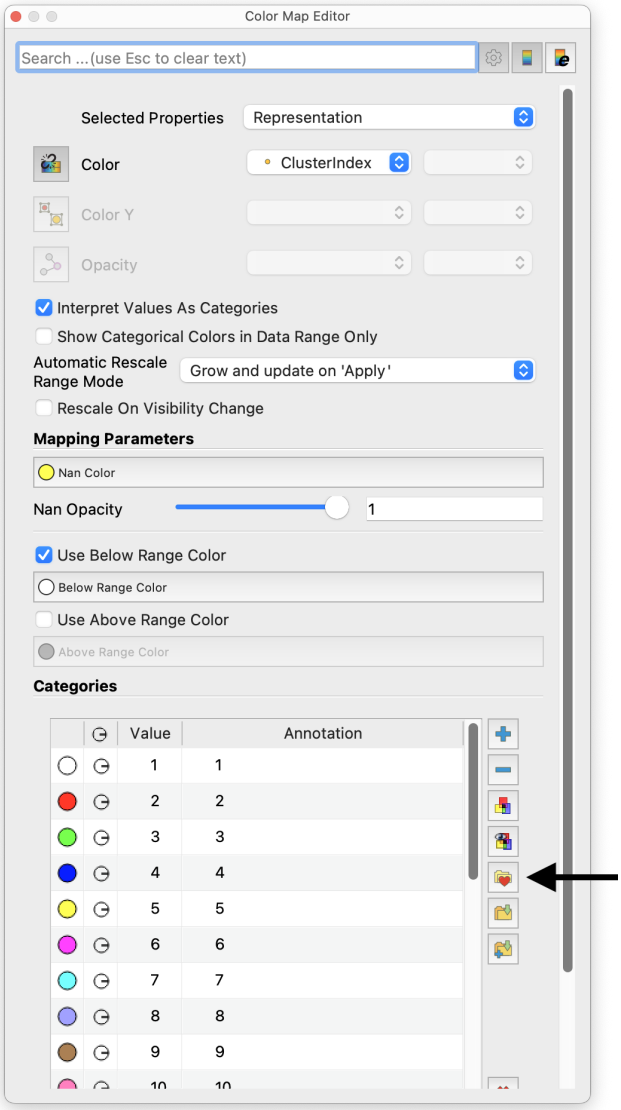


Figure 2: Screenshot of the Color Map Editor window in Paraview v5.13.2, with a black arrow pointing the "Choose Preset" button. (redo with paraview v5.4)

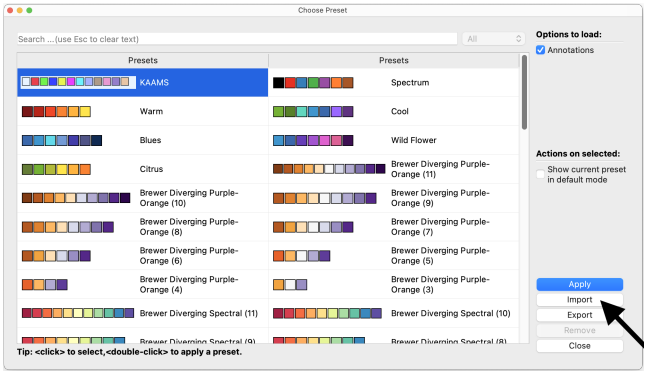


Figure 3: Screenshot of the Choose Preset window in Paraview v5.13.2, with a black arrow pointing the "Import" button. (redo with paraview v5.4)

2.4 Using a python function defined in another file (!!)

To use in a python script (e.g. "main.py") a function that is defined in a separate script (e.g. "utilitaries.py"), first place the two files in the same folder. Then import the content of the secondary file inside your main script using Python *import* statement. For example, if your secondary file is named "utilitaries.py", put the following line at the beginning of your main script to make all the functions defined inside "utilitaries.py" directly accessible :

```
import utilitaries
```

3 Visualizing data

3.1 Visualizing 3D images stored in .tiff files (default use of Paraview)

Open data file

To open a 3D image contained in a .tiff (or .tif) file with Paraview, you can either :

(i) right-click on the file and select the action "Open with > Paraview";

(ii) open a Paraview window and use the "File > Open..." menu in the menu bar;

(iii) open a Paraview window and drag-and-drop the file to open inside the window.

In the first case, Paraview will open a popup window asking you to select the appropriate reader to use for this file (see Figure 4). Please select "TIFF Series Reader".

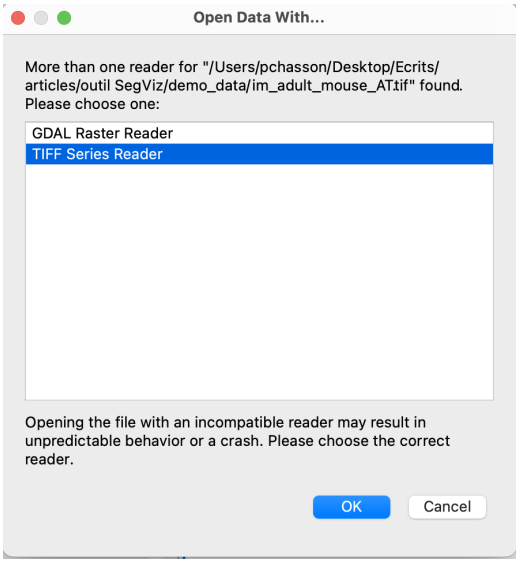


Figure 4: Screenshot of the "Open Data With..." popup window opened by Paraview for .tiff files. (redo with filepath only to Desktop)

Display data

The file should now appear in the "Pipeline Browser" panel of the Paraview window, and its various modifiable properties be displayed in the "Properties" panel (see Figure 5). Click on the "Apply" button at the top of the "Properties" panel to create the display. Alternatively, if the pixels of your image are not cubics, e.g. because the imaging setup does not have the same resolution in the z-direction than in the x and y ones, you can tick the "Use Custom Data Spacing" in the same panel and indicate the relative side-lengths of the pixels in the text boxes below before clicking on "Apply". See Figure 5 for an example with the file `im_adult_mouse_AT.tif` of the folder `demo_data`, whose pixels are 18.4µm ×18.4µm ×10µm wide.

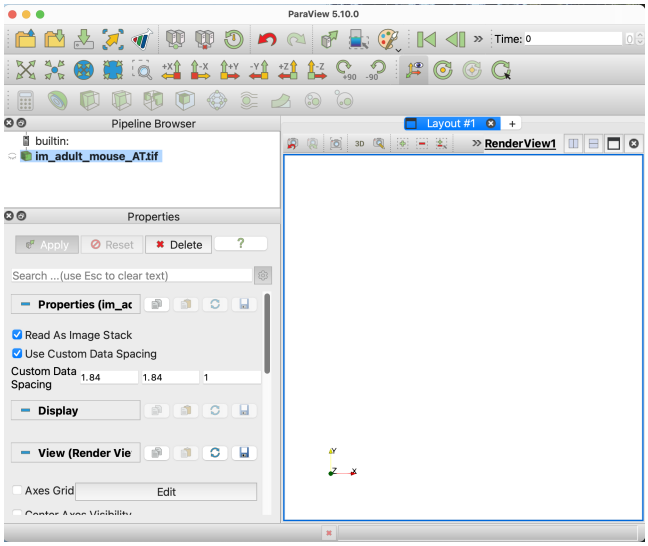


Figure 5: Screenshot of Paraview main window immediately after loading the file `demo_data/im_adult_mouse_AT.tif`.

Change default display style

By default, Paraview use the outline representation, which does not display anything if the content of the file is a binary or labeled image. Use the "Representation" drop-down menu which now appears in the "Properties" panel to select a more appropriate display, for example "Volume" (Figure 6).

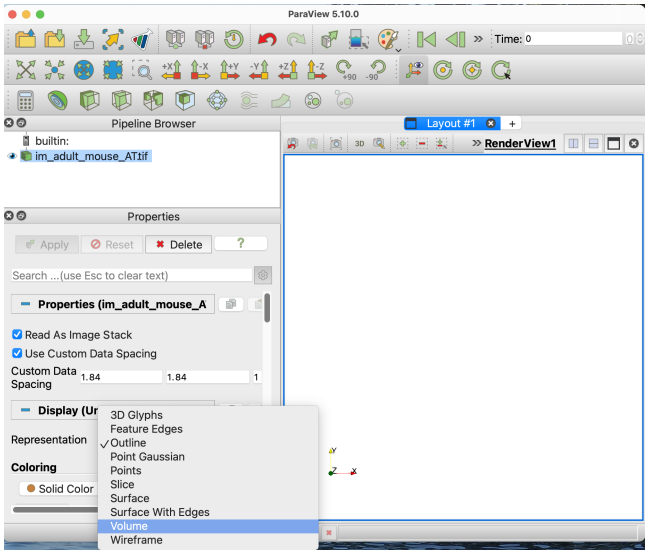


Figure 6: Screenshot of Paraview main window with open "Representation" drop-down menu in the "Properties" panel.

The data should now appear in the "Layout" panel of the Paraview window (see Figure 7). The view angle can be manipulated with the mouse. The default color map can be changed using the Color Map Editor (accessible via the "View > Color Map Editor" entry in the menu bar or the "Edit" button in the "Coloring" section of the "Properties" panel). For a labeled image, a well contrasted color map like the Rainbow Desaturated preset may be more appropriate.

Note that Paraview can not apply categorical (or discrete) color maps to .tiff images. A workaround is to apply to the image a "Threshold" filter with lower threshold 1 (to ignore background) and upper threshold equal to the maximum value in the image (automatically identified by Paraview), then render the thresholded data instead of the original.

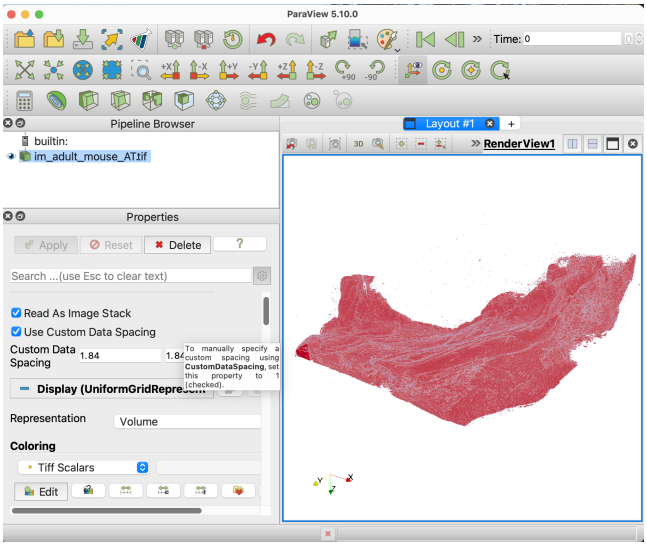


Figure 7: Screenshot of Paraview main window with the image `demo_data/im_adult_mouse_AT.tif` displayed in volume using default color map.

3.2 Visualizing datasets stored in .csv files

Open data file

To open a dataset contained in a .csv file with Paraview, you can either :

(i) right-click on the file and select the action "Open with > Paraview";

(ii) open a Paraview window and use the "File > Open..." menu in the menu bar;

(iii) open a Paraview window and drag-and-drop the file to open inside the window.

In any case, Paraview will open a popup window asking you to select the appropriate reader to use for this file (see Figure 8). Please select "CSV Reader".

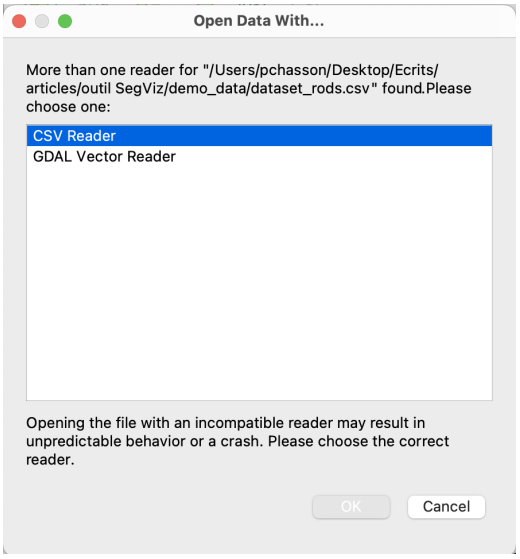


Figure 8: Screenshot of the "Open Data With..." popup window opened by Paraview for .csv files. (redo with file located in Desktop (avoid showing personal filepath) and OK button more readable)

The file should now appear in the “Pipeline Browser” panel of the Paraview window, and its various properties be displayed in the “Properties” panel (see Figure 9). Click on the “Apply” button at the top of the “Properties” panel to load the dataset into a “SpreadSheetView” panel (you can close the corresponding panel to gain some space, without losing the data).

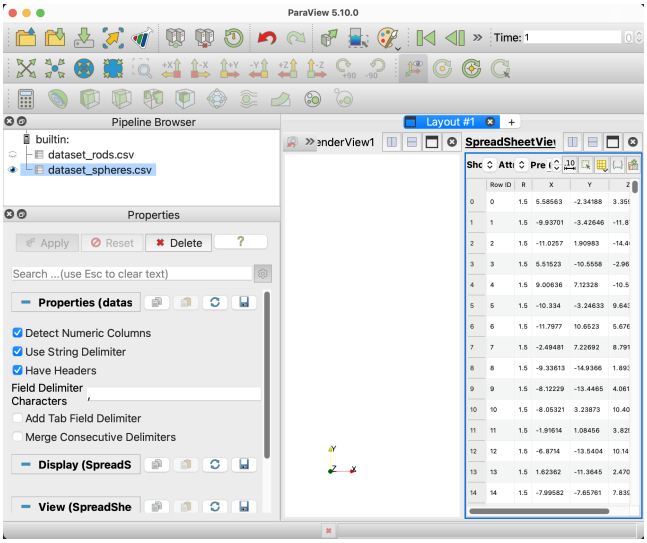


Figure 9: Screenshot of Paraview main window immediately after loading the file `demo_data/dataset_spheres.csv`.

Apply a macro to a dataset

To apply a macro to a dataset, first select this dataset by clicking on its name in the “Pipeline Browser” panel. If the macro is expected to produce a visual, also make sure that the active visualization panel (outlined in blue) is the one where you want this visual to appear. You can activate the desired panel by clicking on it.

You can now apply the desired macro by clicking on its name in the “Macros” drop-down menu of Paraview menu bar (see Figure 1).

Apply macro Sphereviz

The macro **Sphereviz** take as input a dataset containing at least the four following properties (formatted as column header in the first line of the .csv file) : “X”, “Y”, “Z” and “R” pointing respectively the x-, y- and z-coordinate of the object center and its radius.

Give more detailed info about the structure of the .csv files ?

If the dataset also contains a property “ClusterIndex”, each glyph will be colored according to this property using the custom categorical (i.e. discrete) colormap “sashatrubetskoy” (if it is loaded in the Paraview setup) or the default categorical colormap “KAAMS”. Otherwise all the glyphs will be white.

Applying the macro **Sphereviz** to the dataset `demo_data/dataset_spheres.csv`, which does not contain a “ClusterIndex” column, will produce the visual displayed in Figure 10.A.

Applying it to the dataset `demo_data/dataset_spheres_seg_periodic.csv` will produce the visual displayed in Figure 10.B.
This second dataset has been segmented assumingthe spatial domain had periodic boundaries (see corresponding paragraph in section 4.4 below for an explanation). In order to better visualize the shape of the resulting clusters, one can use the translated coordinates of the objects instead of their original coordinates to achieve the visual displayed in Figure 10.C (see section 4.4.3 for more informations about these coordinates). To do this, first click on the “Spheres DataTable” item in the “Pipeline Browser” panel. Then, in the “Properties” panel, change the value of the “X Column”, “Y Column” and “Z Column” properties respectively from X, Y and Z to X_cluster, Y_cluster and Z_cluster (see Figure 10.D).

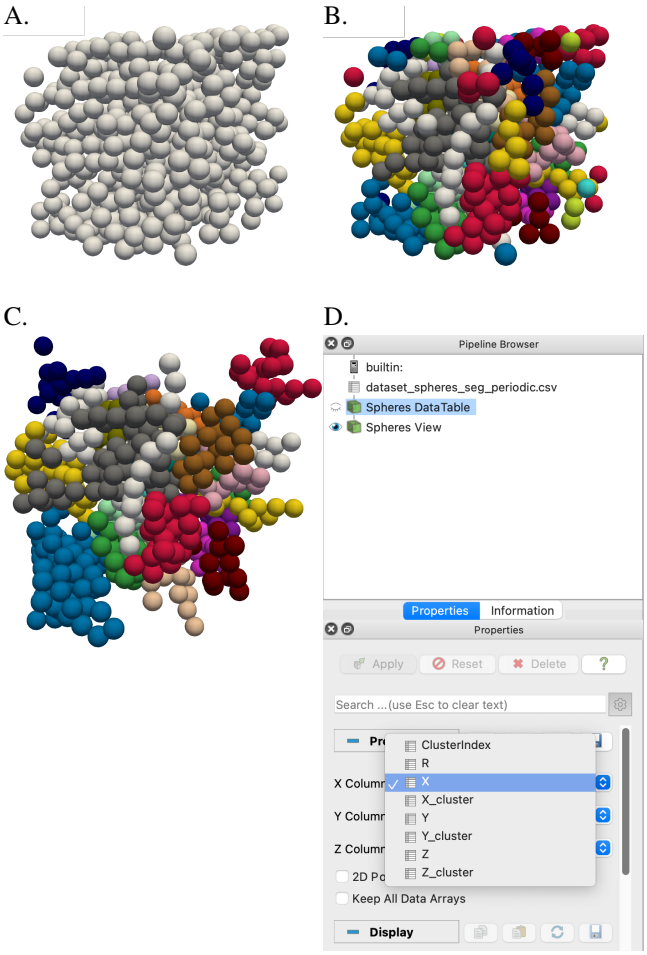


Figure 10: **A.** Screenshot of the visual generated by the macro **Sphereviz** applied to the dataset `demo_data/dataset_spheres.csv`. **B and C.** Screenshots of the visuals generated by the macro **Sphereviz** applied to the dataset `demo_data/dataset_spheres_seg_periodic.csv` using either the objects original coordinates (B) or their translated coordinates (C). **D.** Screenshot of Paraview “Properties” panel with open “X Column” drop-down menu.

Apply macro Rodviz

The macro **Rodviz** take as input a dataset containing at least the seven following properties (formatted as column header in the first line of the .csv file) : “X”, “Y”, “Z”, “wX”, “wY”, “wZ”, “L” and “R” pointing respectively the x-, y- and z-coordinate of the object’s center, the x-, y- and z-component of its unitary directional vector, its length and its radius.
If the dataset also contains a property “color”, each glyph will be colored according to this property using the default “CoolToWarm” colormap. Otherwise all the glyph will have a dark-grey color.

Applying the macro **Rodviz** to the dataset `demo_data/dataset_rods.csv`, which does not contain a “color” column, will produce the display presented in Figure 11.A.

Applying it to the dataset `demo_data/dataset_rods_colored.csv` will produce the display presented in Figure 11.B. To color the objects using another property of the dataset, select the wanted property in the list-menu of the “Coloring” section in the “Properties” panel.

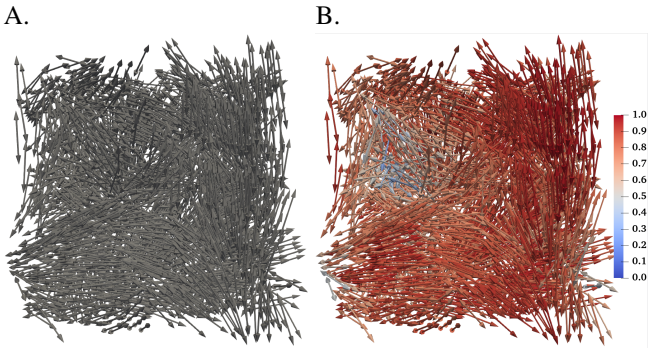


Figure 11: **A.** Screenshot of the visual generated by the macro **Rodviz** applied to the dataset `demo_data/dataset_rods.csv`. **B.** Screenshot of the visual generated by the macro **Rodviz** applied to the dataset `demo_data/dataset_rods_colored.csv`

4 Segmenting data

The examples provided in this chapter are illustrated using our visualization tool (see chapter 3 above). Considering the difficulty of apprehending 3D data, the examples of the first sections are based on simplified dummy data generating simple visuals. Examples based on real data are provided in section 4.7.

4.1 Getting started (!!)

- Dans l'intro, mentionner les input ET output dès le premier paragraphe.
- give beginning of python script : from WatershedSegmentation import WatershedSegmentation
- Typical example of call (default param)

The main function of the file "WatershedSegmentation.py", described in this section, is called **WatershedSegmentation**. It takes as input a parameter describing the type of data to treat (3D binary image or dataset of spherical objects), the data in question and a number of optional parameters allowing the user to control the output format and to fine-tune the segmentation algorithm. All these parameters are listed in table 1 and their use is detailed in the following sections.

Name	Description	Python type	Default value	Used if datatype is "image"	Used if datatype is "dataset"
`datatype`	Type of the input data	*string*	mandatory parameter		
`InputSpheres`	Set of spherical objects to clusterize.	*string* or *numpy.ndarray* or *pandas.DataFrame*	mandatory parameter		
`savefile`	Name of the output file(s).	*string*	"data_seg"	x	x
`return_map`	Whether to return the segmentation map.	*bool*	False		x
`header`	Header of the column containing the clusterization result in the output file.	*string*	"ClusterIndex"		x
`header_per`	Suffix for the header of the three columns containing the translated coordinates in the output file.	*string*	"_cluster"		x
`InputRods`	Set of rod-like objects acting as cluster separators.	*string* or *numpy.ndarray* or *pandas.DataFrame*	None	x	x
`resolution`	Image resolution, in pixel per radius of the smallest object.	*int*	10		x
`xmax`, `ymax`, `zmax`	Half-length of the data spacial domain in each direction.	*float*	None		x
`dil_coeff`	Scaling applied to the spherical objects before segmentation.	*float*	1.0		x
`pixelsize`	Length of the pixels in the three directions.	*list* of three *floats*	[1.0, 1.0, 1.0]	x	
`smooth_coeff`	Smoothing applied to the distance map before segmentation.	*float*	1.0	x	x
`MinSize`	Minimal size of a valid cluster (expressed in pixels if datatype is "image" and in number of objects if datatype is "dataset").	*int*	1	x	x
`Periodic Boundary Condition`	Whether the borders of the computational domain are considered periodic.	*bool*	False		x

Table 1: Summary of the parameters of the function **WatershedSegmentation**.

4.2 Input data

4.2.1 `datatype` : type of the input data

The parameter `datatype` indicates whether the data to segment is in the form of a binary image (`datatype`="image") or a dataset of spherical objects (`datatype`="dataset").

This parameter is mandatory. Python type is *string*, possible values are "image" and "dataset".

4.2.2 `InputSpheres` : data to segment

The user must provide the data to segment via the parameter `InputSpheres`.

If `datatype` is "image" then `InputSpheres` must be a 3D binary-convertible image or a path to such an image, using one of the following formats :

(i) A *numpy.ndarray* of dimension 3 and boolean type.

(ii) A *numpy.ndarray* of dimension 3 and scalar type (i.e. with `dtype` equal to "i", "u" or "f"). The content of this array will be converted to boolean using the function **astype('bool')** and a warning will be issued to the user.

(iii) A *string* with the path to a .tiff (or .tif) file containing a 3D binary or grayscale image, which will be read to a *numpy.ndarray* using the functions **PIL.Image.open** and **PIL.ImageSequence.Iterator**. If the image is grayscale, it will be converted to binary using the function **astype('bool')** on the resulting array. The folder `demo_data` contains two examples of such a file, named `im_spheres.tiff` and `im_adult_mouse_AT.tif`.

If `datatype` is "dataset" then `InputSpheres` must be a dataset of spherical objects or a path to such a dataset, using one of the two following formats :

(i) A *pandas.DataFrame* with at least two rows (i.e. describing at least two objects, otherwise there is nothing to clusterize) and at least four columns with headers "X", "Y", "Z" and "R" containing respectively the x-, y- and z-coordinate of the object's center and its radius. The dataset may contain any number of additional columns.

(ii) A *string* with the path to a .csv file containing the data, which will be read to a *pandas.DataFrame* using the function **pandas.read_csv**. The file must use the default csv format, that is comma ',' for column separator and dot '.' for decimal point. The first line must contain column headers among which "X", "Y", "Z" and "R" pointing respectively the x-, y- and z-coordinate of the objects center and their radius. The order of the columns does not matter and the file may contain any number of additional columns. The folder `demo_data` contains an example of such a file, named `dataset_spheres.csv`.

This parameter is mandatory. Python type is either *string*, *numpy.ndarray* or *pandas.DataFrame*.

4.2.3 `InputRods` : separators for the segmentation

The function **WatershedSegmentation** may be provided, via the parameter `InputRods`, with cluster separators to help segment clusters in the input data.

If `datatype` is "image" then `InputRods` must be a binary image or a path to such an image, using one of formats described in section 4.2.2 above. This image will be used as a negative mask applied to the image provided via `InputSpheres` before the actual segmentation process.

If `datatype` is "dataset" then `InputRods` must be a set of rod-like (or spherocylindrical) objects. These objects will be used as masks during the creation the binary image : that is, pixels located inside a rod-like object will be considered as background even if they are also inside a spherical object. This dataset must be in one of the two following formats :

(i) A *pandas.DataFrame* with at least eight columns with headers "X", "Y", "Z", "wX", "wY", "wZ", "L" and "R" containing respectively the x-, y- and z-coordinates of the object center, the x-, y- and z-components of its unitary directional vector, its length and its radius. The dataset may contain any number of additional columns.

(ii) A *string* with the path to a .csv file containing the data, which will be read to a *pandas.DataFrame* using the function **pandas.read_csv**. The file must use the default csv format, that is comma ',' for column separator and dot '.' for decimal point. The first line must contain column headers, among which "X", "Y", "Z", "wX", "wY", "wZ", "L" and "R" pointing respectively the x-, y- and z-coordinate of the object's center, the x-, y- and z-component of its unitary directional vector, its length and its radius. The order of the columns does not matter and the file may contain any number of additional columns. The folder `demo_data/` contains an example of such a file, named "rods.csv"

Python type is either *string*, *numpy.ndarray* or *pandas.DataFrame*, default value is None.

4.3 Output settings

If `datatype` is "image", the function **WatershedSegmentation** returns a .tiff file (named `savefile`.tiff) containing a labeled image where each pixel contains the index of the cluster to which it pertains. The indexes are continuous starting from 1, with 0 as background.

If `datatype` is "dataset", the function **WatershedSegmentation** returns a .csv file (named `savefile`.csv) containing the original set of spherical objects plus the result of the clusterization process : a column labeled `header` for the cluster index and, if `PeriodicBoundaries` is True, three columns labeled "X`header_per`", "Y`header_per`" and "Z`header_per`" for the translated coordinates of the objects. In addition, if `return_map` is True, the function will returns a .tiff file (named `savefile`.tiff) containing the segmentation map.

4.3.1 `savefile` : name of the output file(s)

The parameter `savefile` enables the user to specify the name, without extension, of the output file(s).

Python type is *string*, default value is "data_seg".

4.3.2 `return_map` : whether to return the segmentation map

If `datatype` is "dataset", the parameter `return_map` enables the user to ask for the segmentation map to be saved to as a labeled image in a .tiff file, in addition to the .csv file containing the segmented dataset. The name of both files will be taken from the parameter `savefile`.

Python type is *bool*, default value is False.

4.3.3 `header`, `header_per` : labels for the segmentation result in the output file

If `datatype` is "dataset", the parameters `header` and `header_per` enable the user to specify the label of the column(s) containing the result of the segmentation process in the output .csv file.

Parameter `header` is used for the column containing the cluster index of each object. If `PeriodicBoundaries` is True, the output .csv file will also contain the translated coordinates of the objects (see corresponding paragraph in section below). These three columns will be labeled respectively "X"+`header_per`, "Y"+`header_per` and "Z"+`header_per`.

Python type is *string*, default values are "ClusterIndex" and "_cluster".

4.4 Pre-processing : creating an image from a dataset

4.4.1 `resolution` : number of pixels in the diameter of the smallest object

If `datatype` is "dataset", the parameter `resolution` indicates the number of pixels that the radius of the smallest object in the dataset `InputSpheres` will cover in the 3D binary image created for segmentation. In other words, the binary image will be composed of cubic pixels of side-length equal to the radius of this smallest spherical object divided by `resolution`.

The minimal value is 1, which ensures that all objects are represented by at least one pixel in the binary image. Setting `resolution` to a high value ensures that small objects are well outlined, but increases the total size of the image and thus the computational cost. Note that, from the perspective of cluster segmentation, a high resolution is usually necessary only if the clusters are very close to each other. An example of such a case is displayed in Figure 12. The results in this figure can be reproduced using the following script :

```
import WatershedSegmentation as ws
ws.WatershedSegmentation("dataset",
  ↪  "demo_data/resolution.csv", resolution=1,
  ↪  savefile="resolution1")
ws.WatershedSegmentation("dataset",
  ↪  "demo_data/resolution.csv", resolution=5,
  ↪  savefile="resolution5")
```

and applying the Paraview macro **Sphereviz** (see section 3.2) to the files resolution1.csv and resolution5.csv thus produced.

Python type is *int*, default value is 5.

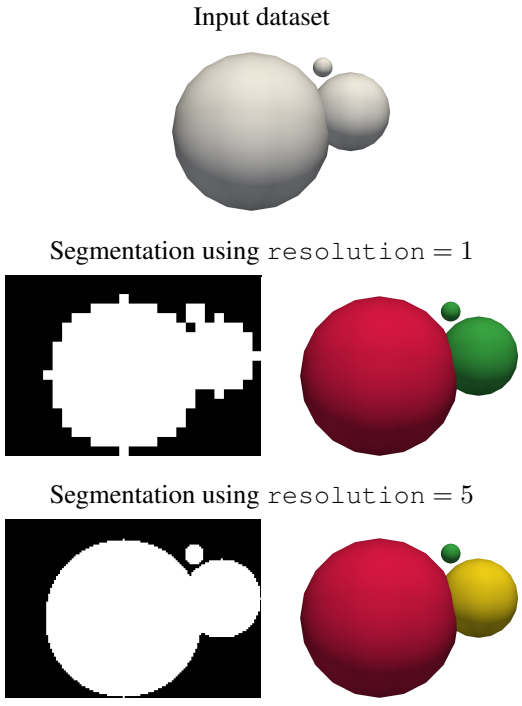


Figure 12: Illustration of the influence of the parameter `resolution` over the segmentation result. **Left column :** One slice of the 3D binary image created by the segmentation tool (objects appear in white and backgound in black). **Right column :** 3D visualization of the segmented dataset using Paraview (all objects pertaining to the same cluster have the same color).

4.4.2 `xmax`, `ymax`, `zmax` : side-length of the computation domain

If `datatype` is "dataset", the algorithm assumes the input data to be contained in a 3D symmetrical domain $[-\texttt{xmax}, \texttt{xmax}] \times [-\texttt{ymax}, \texttt{ymax}] \times [-\texttt{zmax}, \texttt{zmax}]$ and creates a 3D binary image spanning this whole domain.

If the user did not provide a value for one of the half-length parameters `xmax`, `ymax` and `zmax`, the algorithm will estimate the missing value(s) as the smallest value(s) allowing the domain to enclose all the objects referenced in the dataset `InputSpheres` :

$$\begin{aligned}\texttt{xmax} &= \max(|X_i| + R_i \quad \forall i \in \texttt{InputSpheres})\\ \texttt{ymax} &= \max(|Y_i| + R_i \quad \forall i \in \texttt{InputSpheres})\\ \texttt{zmax} &= \max(|Z_i| + R_i \quad \forall i \in \texttt{InputSpheres})\end{aligned}$$

Figure 13 illustrates this process.

Note that user-provided values are really needed only if `PeriodicBoundaries` is True, to find out which part of the objects cross the border of the domain (see next section).

Python types are *float*. Default values are None, leading to automatic computation.

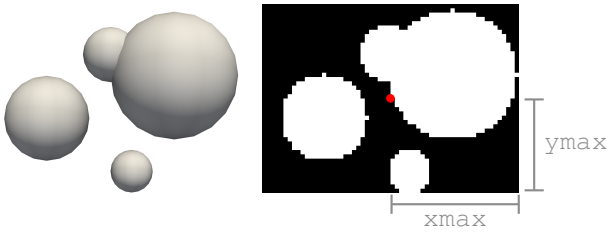


Figure 13: Example of automatic computation of the domain size from the input data. **Left :** 3D visualization of the input dataset using Paraview **Right :** One slice (along the z-axis) of the 3D binary image created by the segmentation tool (objects appear in white and backgound in black). The center point $(0, 0)$ is marker by a red dot and the half-lengths `xmax` and `ymax` are indicated with gray lines.

4.4.3 `PeriodicBoundaries` : whether the input dataset assume periodic boundaries

If the input dataset is derived from a mathematical model including periodic boundary conditions (i.e. any object exiting the domain from one side re-enters it from the opposite side), then the segmentation process should also include periodic boundaries. This is done by setting the parameter `PeriodicBoundaries` to True.

Note that, in this case, the user must provide the exact size `xmax`, `ymax`, `zmax` of the computational domain (see previous section) : if these parameters are set to the default, automatic computation then no objects will cross the border and the periodic boundaries will have no effect. For this reason, if `PeriodicBoundaries` is True and one of the half-length `xmax`, `ymax` or `zmax` is missing, a warning will be issued to the user. Figure 14 illustrates how the parameters `xmax`, `ymax`, `zmax` and `PeriodicBoundaries` interact.

- **A**. When all the domain's side-lengths are automatically computed, the result of the segmentation is the same whether `PeriodicBoundares` is True or False, with three clusters identified.
- **B**. With an appropriate user-provided value of `xmax` and automatic computation of `ymax`, two of the previously identified clusters are merged through the periodic x-boundary.
- **C**. With appropriate user-provided values of `xmax` and `ymax`, all previously identified clusters are merged through the periodic boundaries.

The results in this figure can be reproduced using the following script :

```
import WatershedSegmentation as ws
ws.WatershedSegmentation("dataset",
  ↪  "demo_data/domain_size.csv",
  ↪  savefile="domain_size_auto")
ws.WatershedSegmentation("dataset",
  ↪  "demo_data/domain_size.csv", xmax=4.2,
  ↪  savefile="domain_size_xmax4.2")
ws.WatershedSegmentation("dataset",
  ↪  "demo_data/domain_size.csv", xmax=4.2, ymax=3,
  ↪  savefile="domain_size_xmax4.2_ymax3")
```

and applying the Paraview macro **Sphereviz** (see section 3.2) to the files domain_size_auto.csv, domain_size_xmax4.2.csv and domain_size_xmax4.2_ymax3.csv thus produced.

Python type is *bool*, default value is False.

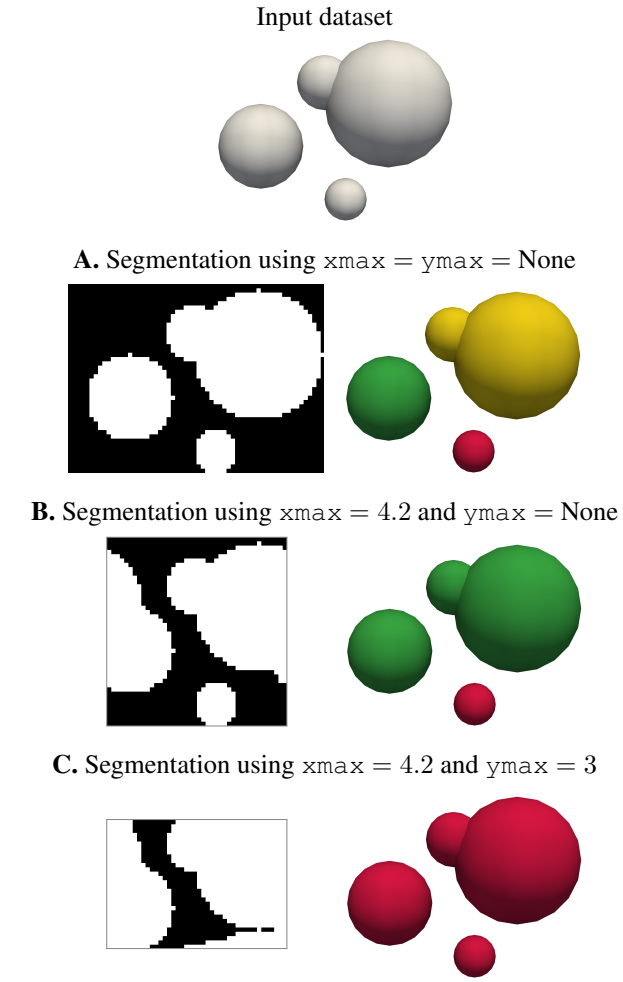


Figure 14: Example of application of periodic boundary conditions to the same data with different domain side-lengths. **Left column :** One slice of the 3D binary image created by the segmentation tool (objects appear in white and backgound in black). For the sake of clarity, a gray frame has been added to the images in rows **B** and **C**. **Right column :** 3D visualization of the segmented dataset using Paraview (all objects pertaining to the same cluster have the same color).

4.4.4 `dil_coeff` : scaling factor

If `datatype` is "dataset", clustering may be made easier by scaling the size of the spherical objects up or down, that is by multiplying their radius by a factor `dil_coeff` when inserting them in the 3D binary image (see Figure 15 for examples).

For instance, if there are small gaps between objects pertaining to the same cluster, the distance transform will display a local minima at the center of each object and watershed clustering will thus segment the individual objects instead of the cluster (see Figure 15.A). In this case, scaling up the objects by a small factor may fill the holes without unduly connecting separate clusters (see Figure 15.B). This option is especially intended for cases where the user also provides a set of rod-like objects to serve as separators between clusters, as the risk of unwanted cluster merging is then reduced (see Figure 15.C).

If, on the other hand, the user wants to divide a continuous cluster into multiple components based on a notion of objects density, scaling down may allow it by creating holes inside the low density regions (compare Figure 15.D and E). This option is intended for cases where there are large discrepancies in the objects local density.

The results in Figure 15 can be reproduced using the following script :

```
import WatershedSegmentation as ws
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/scaling.csv", savefile="scaling1")
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/scaling.csv", dil_coeff=1.2,
   ↪  savefile="scaling1.2")
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/scaling.csv", dil_coeff=1.2,
   ↪  InputRods="demo_data/scaling_rod.csv",
   ↪  savefile="scaling1.2_withsep")
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/scalingdown.csv",
   ↪  savefile="scalingdown1")
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/scalingdown.csv", dil_coeff=0.9,
   ↪  savefile="scalingdown0.9")
```

and applying the Paraview macro **Sphereviz** (see section 3.2) to the files thus produced.

As a rule, it is best to keep the factor `dil_coeff` close to 1 as values too high or too low are likely to produce unwanted results.

Python type is *float*, default value is 1 (i.e. no scaling).

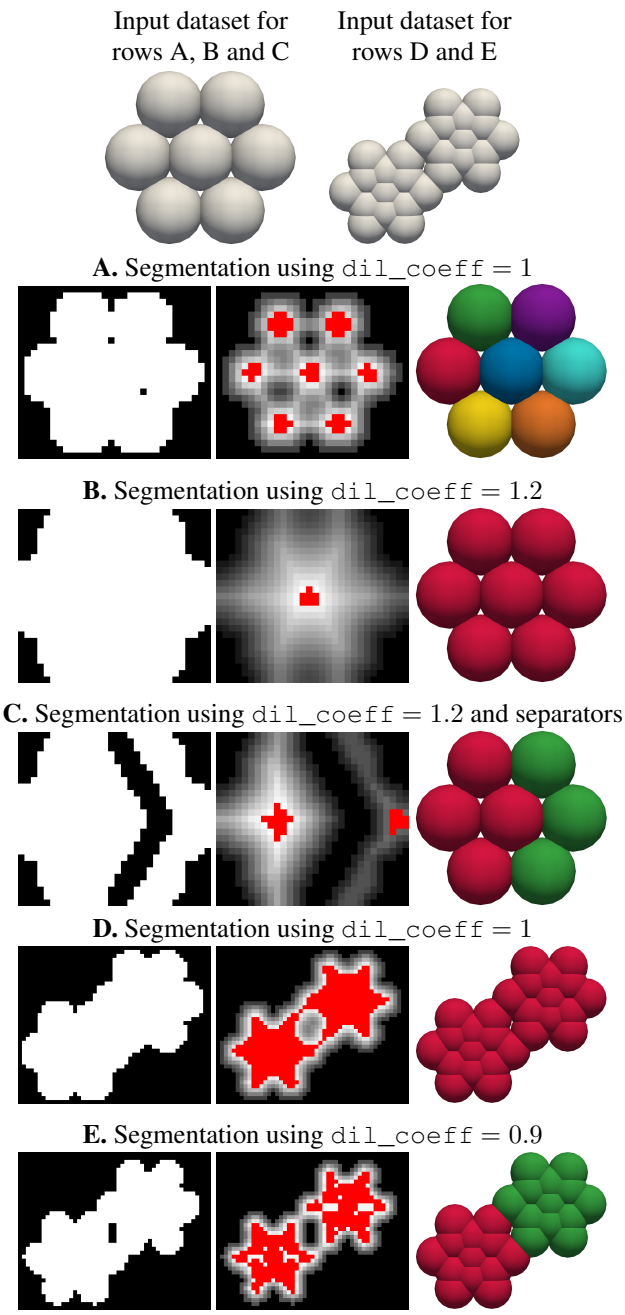


Figure 15: Example of scaling up a dataset. **Left column :** One slice of the binary image created by the segmentation tool (objects appear in white and backgound in black). **Middle column :** Same slice of corresponding distance map (with local minima indicated in red). **Right column :** 3D visualization of the segmented dataset using Paraview (all objects pertaining to the same cluster have the same color).

4.5 Watershed settings

4.5.1 `pixelsize` : side-lengths of the pixels in the input image

If `datatype` is "image", the parameter `pixelsize` indicates the relative side-length of the pixels of the image in the x, y and z directions. Unit is implicit and the same than the one of parameter `smooth_coeff` (see next section). This option is intended to account for pixel anisotropy in 3D microscopy images, which usually have a lower resolution in depth (i.e. between two slices) than along the slicing the planar, meaning that the pixels have a much larger side-length in the z direction than in the two others.

Python type is *list* of three *floats*, default value is $[1.0, 1.0, 1.0]$.

4.5.2 `smooth_coeff` : region smoothing (or peak filtering)

The watershed algorithm segment clusters in a binary image based on seeds. These seeds are usually defined as the local minima in the euclidean distance transform of the image. However, taking all local minima into account is likely to result in over-segmentation, since any indentation in a cluster's surface may generate additional local minima in the distance map. Hence, it is common to filter such shallow minima by applying to the distance transform a smoothing function. This segmentation tool implements a modified version of the h-minima smoothing function, designed to reproduce the behaviour of Matlab's imhmin function : it erases all local minima whose depth, with respect to their surroundings, is less than the distance `smooth_coeff`.

Figure 16 illustrates the influence of this smoothing step. In each case, whether the data are over-segmented or not depends on the user's objective.

A. The non-smoothed euclidiean distance transform presents local minima at the center of each object, leading to segmentation of 13 isolated objects.

B. Smoothing the euclidean distance transform by a factor `smooth_coeff`$= 1$ reduces the number of local minima to 7. Note how the remaining local minima span more pixels due to the smoothing operation.

C. Smoothing the euclidean distance transform by a factor `smooth_coeff`$= 5$ reduces the number of local minima to 1, leading to the segmentation of one cluster containing all the objects.

The results in this figure can be reproduced using the following script :

```
import WatershedSegmentation as ws
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/smoothing.csv", smooth_coeff=0,
   ↪  savefile="smoothing0")
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/smoothing.csv",
   ↪  savefile="smoothing1")
ws.WatershedSegmentation("dataset",
   ↪  "demo_data/smoothing.csv", smooth_coeff=5,
   ↪  savefile="smoothing5")
```

and applying the Paraview macro **Sphereviz** (see section 3.2) to the files smoothing0.csv, smoothing1.csv and smoothing5.csv thus produced.

Note that the distance `smooth_coeff` is expressed in the same unit than the pixelsize of the image (for more details, see parameter `resolution` in section 4.4 concerning dataset inputs; and parameter `pixelsize` above concerning image inputs).

Python type is *float*, default value is 1.

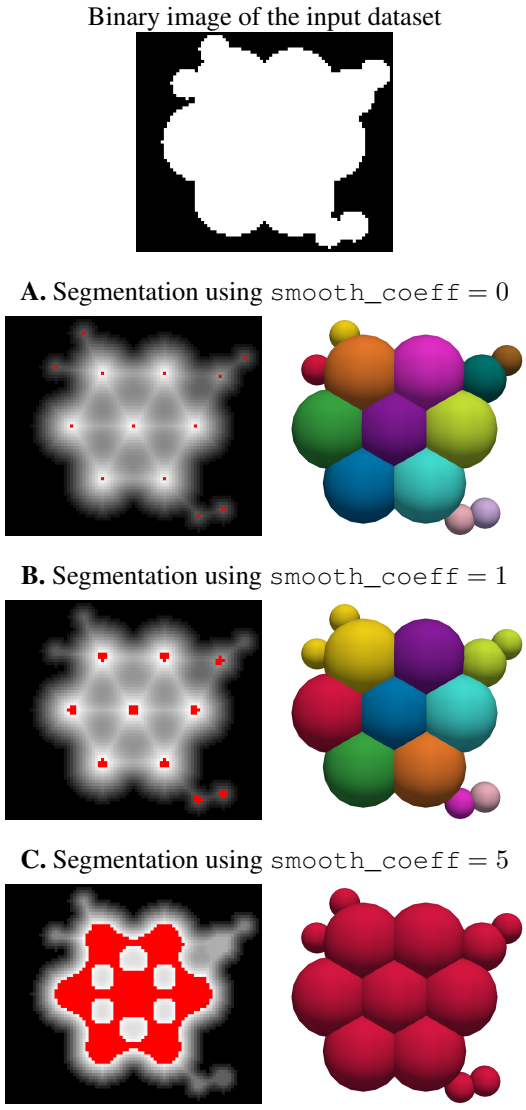


Figure 16: Illustration of the influence of the parameter `smooth_coeff` over the segmentation result. **Left column :** One slice of the distance map (with local minima indicated in red) corresponding to the input dataset. **Right column :** 3D visualization of the segmented dataset using Paraview (all objects pertaining to the same cluster have the same color).

4.6 `MinSize` : removing small clusters in post-processing

Small clusters may not interest users. Hence, the parameter `MinSize` allows the user to define a minimum size for a cluster to be considered valid. If `datatype` is "image", this parameter is expressed in number of pixels in the cluster. If `datatype` is "dataset", it is expressed in number of objects in the cluster. Appropriate values may vary widely depending on the user's purpose.

Please note that this option does not affect the segmentation process in itself : it is only a post-processing step which erase

small clusters. It is mainly intended for users who plan to do statistical computation on the segmented clusters and do not want to take into account small, miscellaneous clusters. It *does not* merge multiple small clusters into a large cluster or blend a small cluster into a neighbouring large cluster (such results could however be achieved using the parameter `smooth_coeff`, see corresponding section above).

If `datatype` is "image", invalid clusters will be removed from the segmented image (i.e. blended into the background) and the remaining clusters will be numbered continuously starting from 1 (with 0 as the background value). If `datatype` is "dataset", objects pertaining to invalid clusters will get the cluster index -1, while valid clusters are numbered continuously starting from 0 to be consistent with Python indexing.

Python type is *int*, default value is 1 (meaning that all clusters will be valid).

4.7 Practical examples (!!)

The following section present some example of use of the function **WatershedSegmentation**. The data used in these examples can be found in the folder demo_data.

4.7.1 Image segmentation

To segment a binary image using the default setting, use the following instruction :

```
WatershedSegmentation(datatype='image', InputSpheres="demo_data/im_spheres.tiff")
```

This should produce a file "data_seg.tiff" identical to the file "demo_data/im_spheres_seg_default.tiff". Opening this file with Paraview should produce the display illustrated in Figure 17.A.

To use another binary image as a separator mask and save the resulting image in a file named "im_spheres_seg_withmask.tiff", use the following instruction :

```
WatershedSegmentation(datatype='image', InputSpheres="demo_data/im_spheres.tiff",
  ↪  InputRods="demo_data/im_rods.tiff", savefile="im_spheres_seg_withmask")
```

Note how the value of the parameter `savefile` does not include the .tiff extension : this is determined automatically from the type of output data. Opening the resulting file with Paraview should produce the display illustrated in Figure 17.B.

If the image pixels are not cubics, e.g. because the imaging setup does not have the same resolution in the z-direction than in the x and y ones, use the parameter `pixelsize` to indicate the relative side-lengths. For example, if one pixel is 18.4µm $\times$18.4µm $\times$10µm wide, use the following instruction :

```
WatershedSegmentation(datatype='image', InputSpheres="demo_data/im_adult_mouse_AT.tif",
  ↪  pixelsize=[1.84,1.84,1])
```

Use the parameter `Minsize` to remove cluster containing less than a given number of pixels :

```
WatershedSegmentation(datatype='image', InputSpheres="demo_data/im_adult_mouse_AT.tif",
  ↪  pixelsize=[1.84,1.84,1], MinSize=10000)
```

Finally, use the parameter `smooth_coeff` to apply a smoothing coefficient to the image and avoid over-segmentation :

```
WatershedSegmentation(datatype='image', InputSpheres="demo_data/im_adult_mouse_AT.tif",
  ↪  pixelsize=[1.84,1.84,1], MinSize=10000, smooth_coeff=8)
```

Opening the file resulting from this last example with Paraview should produce the display illustrated in Figure 17.C.

A. | B. | C

Figure 17: plus tard

4.7.2 Dataset segmentation

To segment a dataset of spherical objects using the default setting, use the following instruction :

```
WatershedSegmentation(datatype='dataset', InputSpheres="demo_data/dataset_spheres.csv")
```

This should produce a file "data_seg.csv" identical to the file "demo_data/dataset_spheres_seg_default.csv". Opening this file with Paraview should produce the display illustrated in Figure 18.A.

To use a dataset of rod-like objects as a separator mask and save the resulting image in a file named "dataset_spheres_seg_withmask.csv", use the following instruction :

```
WatershedSegmentation(datatype='dataset', InputSpheres="demo_data/dataset_spheres.csv",
  ↪  InputRods="demo_data/dataset_rods.csv", savefile="dataset_spheres_seg_withmask")
```

Note how the value of the parameter `savefile` does not include the .csv extension : this is determined automatically from the type of output data. Opening the resulting file with Paraview should produce the display illustrated in Figure 18.B.

A more interesting clustering can be obtained by tuning some parameters :

```
WatershedSegmentation(datatype='dataset', InputSpheres="demo_data/dataset_spheres.csv",
  ↪  InputRods="demo_data/dataset_rods.csv", resolution=3, dil_coeff=1.5, smooth_coeff=0.5)
```

Use the parameter `Minsize` to remove cluster containing less than a given number of objects :

```
WatershedSegmentation(datatype='dataset', InputSpheres="demo_data/dataset_spheres.csv",
  ↪  InputRods="demo_data/dataset_rods.csv", resolution=3, dil_coeff=1.5, smooth_coeff=0.5, MinSize=10,
  ↪  savefile="dataset_spheres_seg_coarse")
```

The segmentation map can be saved in a .tiff file by setting the parameter `return_map` to True :

```
WatershedSegmentation(datatype='dataset', InputSpheres="demo_data/dataset_spheres.csv",
  ↪  InputRods="demo_data/dataset_rods.csv", resolution=3, dil_coeff=1.5, smooth_coeff=0.5, MinSize=10,
  ↪  savefile="dataset_spheres_seg_coarse", return_map=True)
```

Finally the segmentation process can assume periodic boundary condition on the border of the spatial domain by setting `PeriodicBoundaryCondition` to True. In that case, the user must also provide the accurate half-length of the spatial domain in each direction via the parameters `xmax`, `ymax` and `zmax` :

```
WatershedSegmentation(datatype='dataset', InputSpheres="demo_data/dataset_spheres.csv",
  ↪  InputRods="demo_data/dataset_rods.csv", resolution=3, dil_coeff=1.5, smooth_coeff=0.5, MinSize=10,
  ↪  PeriodicBoundaries=True, xmax=15, ymax=15, zmax=15, savefile="dataset_spheres_seg_periodic")
```

A. | B. | C

Figure 18: plus tard

4.8 Brief description of the algorithm

The function **WatershedSegmentation** takes as input a parameter describing the type of data to treat (3D binary image or dataset of spherical objects), the data in question and a number of optional parameters listed in table 1 (see sections **??** and **??** above for more details).

If the input data is a 3D binary image, the algorithm will call the sub-function **ClusterizeBinaryImage** to segment the image into different region using the watershed transformation and write the resulting segmentation map into a .tiff file as a labeled 3D image. You can type the following instruction in your Python executor to get more info on this sub-function and its specific parameters :

```
help ClusterizeBinaryImage
```

If the input data is a dataset of spherical objects, the algorithm will call the sub-function **ClusterizeSphereObjects** to group these objects into clusters (based on spatial proximity using the watershed transformation) and write the result of this clusterization into a .csv file. Optionally, the clusterization map can be saved in a .tiff file as a 3D labeled image. You can type the following instruction in your Python executor to get more info on this sub-function and its specific parameters :

```
help ClusterizeSphereObjects
```

Both of the above-mentionned functions rely on the same algorithm for segmenting a 3D binary image into separate regions using the watershed transform. This algorithm is implemented in the function **MapRegionsUsingWatershed**, which can be broken down as follows (the name of the related sub-functions are given in bold text) :

1. Apply euclidean distance transform to create a grayscale map indicating the distance of each white pixel of the initial binary image to the black background.

   ↳ **scipy.ndimage.distance_transform_edt**

2. Filter this distance map to remove shallow local minima.

   ↳ **imhmin**

3. Identify and label local minima in the filtered distance map.

   ↳ **skimage.morphology.local_maxima**

   ↳ **scipy.ndimage.label**

4. Apply watershed algorithm to the filtered distance map, using local minima as seeds.

   ↳ **skimage.segmentation.watershed**

If the input data is 3D binary image, no further processing is required. If it is a dataset instead, then this dataset must be pre-processed into a 3D binary image where objects are white and background is black (using the function **Create3Dbinaryimage**), and the segmentation map must be post-processed to identify the cluster index of each object of the dataset (using the function **ClusterizeFromMap**).

Acknowledgements

Bibliography

[1] P. Chassonnery, J. Paupert, A. Lorsignol, C. Sévérac, M. Ousset, P. Degond, L. Casteilla, D. Peurichard, Fiber crosslinking drives the emergence of order in a three-dimensional dynamical network model. *Royal Society Open Science* **11**, 231456 (2024).

[2] P. Chassonnery, D. Peurichard, S. Haliyo, 3D visualisation and segmentation tools for systems of spherical and rod-like objects. (in preparation).

[3] P. Chassonnery, J. Paupert, A. Lorsignol, C. Sévérac, M. Vigneau, L. Pieruccioni, M. Ousset, P. Degond, L. Casteilla, D. Peurichard, Morphogenesis of adult adipose tissues emerges from simple local mecanical interactions. (in preparation).

[4] J. Ahrens, B. Geveci, C. Law, "ParaView: An End-User Tool for Large Data Visualization" in *Visualization Handbook*, C. D. Hansen, C. R. Johnson, Eds. (Elsevier, 2005), pp. 717–731.

[5] S. Trubetskoy, List of 20 Simple, Distinct Colors. https://sashamaps.net/docs/resources/20-colors/ Accessed: 2024-04-17

[6] F. Meyer, Topographic distance and watershed lines. *Signal Processing* **38**, 113–125 (1994).

[7] page de ref python sur l'usage du watershed ?

[8] ref octave (version open-source de matlab) pour la fonction imhmin ?