# MultivacDisplay 3.1 User's Guide

Vivien Mallet
vmallet@ec-lyon.fr

October 22, 2006

## Contents



## Introduction

MultivacDisplay is a Python package for visualization of Multivac output results. It provides Python functions that may be used in scripts or in interactive mode. It also includes a graphical user interface for image preprocessing.

There have been several packages to display Multivac results, in interactive mode, in batch mode or through a GUI (graphical user interface). MultivacDisplay is the outcome of these previous experiments and from other experiments in other fields. Although the interface will probably evolve, it is very likely that the technical solution (Python, Matplotlib, SciPy, Qt) is perennial. It is therefore worth learning MultivacDisplay and underlying resources. Besides Python and associated tools are open source, increasingly used in the scientific community, and very relevant in many fields.

This user's guide should first help in the installation (Section 1). It explains the way MultivacDisplay may be used to display Multivac results in Section 2. It ends with a section about the GUI (Section 3).

## 1 Installation

### 1.1 Requirements

MultivacDisplay requires:

- Python 2.3 or higher – `http://www.python.org/`;
- NumPy – `http://numpy.scipy.org/`;
- Matplotlib – `http://matplotlib.sourceforge.net/`.

For image segmentation, it is necessary to load image from Python with:

- the Python Imaging Library – `http://www.pythonware.com/products/pil/`.

The GUI requires in addition:

- PyQt – `http://www.riverbankcomputing.co.uk/pyqt/index.php`.

In order to work in an interactive Python shell (like in Matlab or Mathematica), it is recommended to install IPython – `http://ipython.scipy.org/`. Instructions may be found on Matplotlib website in section "Interactive".

For Debian users (Etch and next releases), packages are available in the official repository. Packages will properly be installed with this command (as root):

```
apt-get install python python-qt3 ipython python-numpy python-matplotlib \
python-imaging
```

In order to learn Python, I recommend the following introduction: "Dive into Python", available at `http://diveintopython.org/`. If you want a good overview of Python, read sections 2, 3, 4 and 6. Shorter introductions may be found on the Internet and are enough to use MultivacDisplay. Additional documents may be found at `http://docs.python.org/`.

It is recommended to browse Matplotlib tutorial. It takes a few minutes since Matplotlib has an easy syntax, mostly the same as Matlab.

## 1.2 MultivacDisplay

Once previous packages are installed, one has to uncompress MultivacDisplay:

```
tar -zxvf MultivacDisplay-3.1.tar.gz
```

This creates a directory called `MultivacDisplay`. To check that all works, launch Python (command `python` for shell) from where directory `MultivacDisplay` lies (but not *in* directory `MultivacDisplay`). Then type `import MultivacDisplay` to check that the module can be loaded by Python.

When a module is included, Python searches for it in the local directory and, then, in the paths in the shell variable `$PYTHONPATH` (just like variable `$PATH` is used by Bash or any similar shell). Therefore, to use MultivacDisplay, it is necessary to put it in the local directory or in `$PYTHONPATH`.

To complete the installation, it is recommended to compile the C++ program `extract_configuration` in `MultivacDisplay/talos/`. It is required by program `image`, by the GUI and it is used by a few functions in MultivacDisplay too. The compilation line (launched in `MultivacDisplay/talos/`) may be:

```
g++ -I Talos -o extract_configuration extract_configuration.cpp
```

You may use another compiler providing it is reasonably compliant with the C++ standard.

# 2 Python Interface

## 2.1 Loading Multivac Results

Multivac generates a file that stores the coordinates of the front points (default name: `Curves`), and a file with the number of points stored at each timestep (default name: `CurveLengths`). They are text files stored in the result directory (default directory: `results/`). In this subsection, the results from program `track`, with its default configuration, are used.

MultivacDisplay enables to load the results with the following commands (quoted from a IPython session):

```
In [1]: from MultivacDisplay import *

In [2]: front = load_front("results/Curves", "results/CurveLengths")
```

Note that, in MultivacDisplay, plurals are avoided in function names.

Variable `front` is a list of fronts. Each element `front[i]` is the `i`-th front saved in Multivac results. Element `front[i]` consists of a NumPy 2D-array with all points that discretize the `i`-th front: `front[i][n, 0]` is the abscissa of the `n`-th point of the `i`-th front, and `front[i][n, 1]` is its ordinate.

For instance, the five first points of the third front have the following coordinates:

```
In [3]: front[2][:5] # short notation for front[2][0:5, :]
Out[3]:
array([[ 0.499958,  1.44    ],
       [ 0.5     ,  1.43968 ],
       [ 0.501262,  1.43    ],
       [ 0.50277 ,  1.42    ],
       [ 0.504478,  1.41    ]])
```

All functions are documented. You may therefore print their "docstring" (e.g., `print load_front.__doc__`), or use the command `help` under IPython:

```
In [4]: help(load_front)
Help on function load_front in module MultivacDisplay.display:

load_front(curves_file, curve_lengths_file)
    Loads fronts from Multivac output files.

    @type curves_file: string
    @param curves_file: The path to the file that stores all front points.
    @type curve_lengths_file: string
    @param curve_lengths_file: The path to the file that stores the number of
    points in each front.

    @rtype: list of 2D numpy.ndarray
    @return: All fronts in a list. Each front is represented by a (N x
    2)-array where N is the number of points in the front.
```

## 2.2 Displaying Fronts

Once front have been loaded, possibly modified or created with the proper structure described previously, it is possible to display the fronts with `display_front`:

```
In [5]: display_front(front)
```

You may save it in any format supported by your Matplotlib backend (most likely, at least eps, png and jpg), e.g.:

```
In [6]: savefig("front-color.eps")
```

This results in Figure 1.

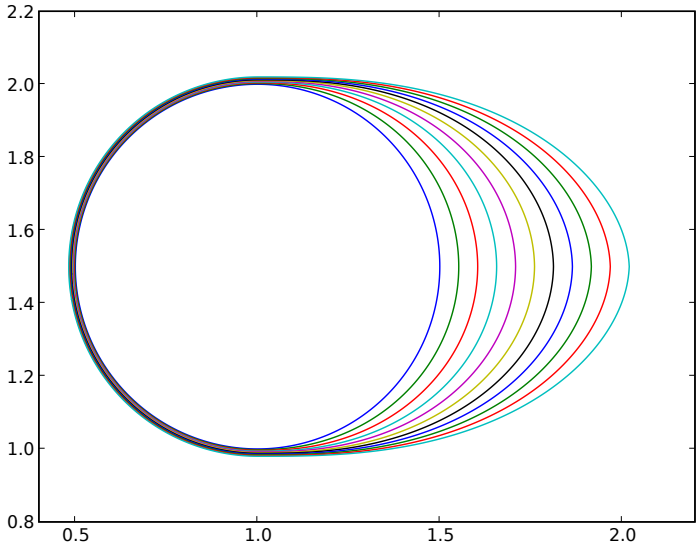



Figure 1: Output fronts of program `track` displayed by function `display_front`.

Function `display_front` accepts optional arguments of function `plot`. For instance, in order to have all fronts displayed in black and with a discontinuous lines, launch:

```
In [7]: clf(); display_front(front, 'k--')
```

Function `clf` clears the figure, and function `display_front` is called with a `plot`-like argument. See Figure 2. Usually one uses black curves with continuous lines, that is, option `'k-'`.

Function `display_front` also accepts all `plot` keyword arguments and the additional keyword argument `dist`. This argument specifies the minimum distance between two distinct front curves at the same timestep. For each timestep, the output of Multivac is a set of points that are on the front. The front may be composed of several distinct curves. To display the curves, `display_front` plots segments whose bounds are points on the front. But points from two distinct curves should not be linked. Distinct curves are detected where two points of the front are "too far" from each other. In practice, it is recommended to set the minimum distance between two curves to $\sqrt{\Delta x^2 + \Delta y^2}$ where $\Delta x$ and $\Delta y$ are the space steps along $x$ and $y$ respectively. A close approximation to this number is usually good enough.

Figure 3 shows an example with the distance set properly or badly. After having run Multivac program `merge_and_island`, commands in IPython are:

```
In [8]: front = load_front("results/Curves", "results/CurveLengths") # Loads new results.
```

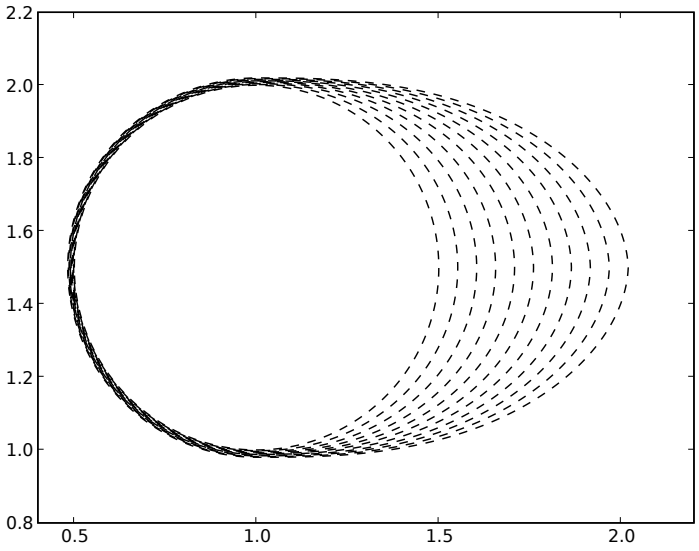



Figure 2: Output fronts of program `track` displayed by function `display_front` with option `'k--'`.

```
In [9]: clf(); display_front(front, 'k-', dist = 1.) # Wrong distance.

In [10]: clf(); display_front(front, 'k-', dist = 0.01)
```

One may select the fronts to be displayed. Below are examples:

```
In [11]: display_front(front[0], 'k-', dist = 0.01) # First front.

In [12]: display_front(front[-1], 'k-', dist = 0.01) # Last front.

In [13]: display_front(front[::2], 'k-', dist = 0.01) # One front each two.

In [14]: display_front(front[10:], 'k-', dist = 0.01) # Excludes the first 10 fronts.
```

Of course, all Matplotlib function are still available. For example, you can still modify the axis extent (`xlim` and `ylim`), add a title (`title`), put a legend (`legend`), etc.

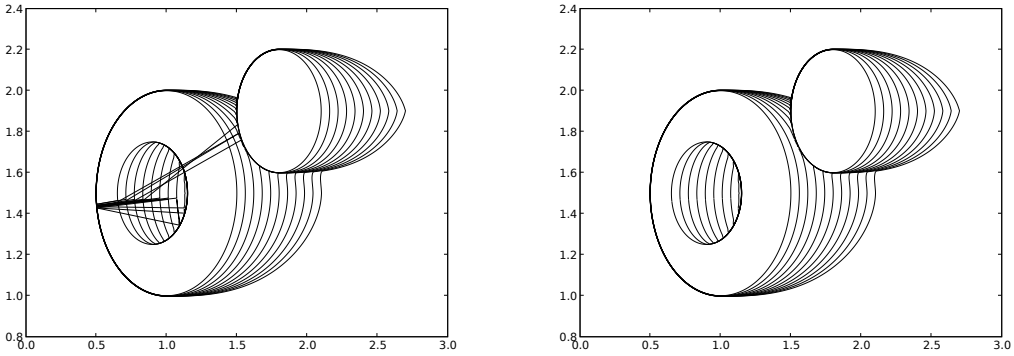
### 2.3 Images in the Background

Multivac may be used for image segmentation. It may be useful to put an image in background. For this, with the Python Imaging Library installed, one may use function `load_image`. This function takes the path of a file as argument (png file, for instance) and returns a NumPy array ready to be used in Matplotlib.

Below is an example:

```
In [15]: image = load_image("/home/vivien/src/multivac/chan-vese/fire.jpg")

In [16]: clf(); imshow(image)
```




(a) With a wrong distance.

(b) With the right distance.

Figure 3: Results displayed by `display_front` with fronts composed of multiple curves. The minimum distance between two curves should be properly set to avoid segments between distinct curves. The keyword argument of `display_front` should be to $\sqrt{\Delta x^2 + \Delta y^2}$ where $\Delta x$ and $\Delta y$ are the space steps along $x$ and $y$ respectively.

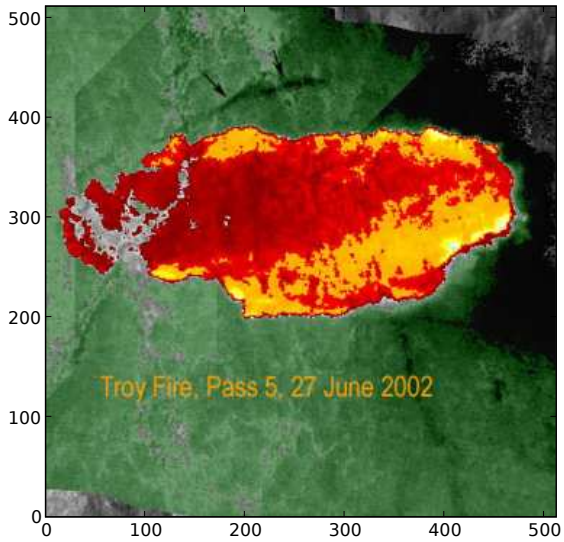



Figure 4: Image loaded with `load_image` displayed by `imshow`.

This results in Figure 4.

Fronts may be displayed on top of the image with `display_front`. For instance, one may show the initial and the final fronts of an image segmentation experiment:

```
In [17]: display_front(front[0], 'w-', linewidth = 2.5)

In [18]: display_front(front[-1], 'w-', linewidth = 2.5)
```

Figure 5 shows the results.

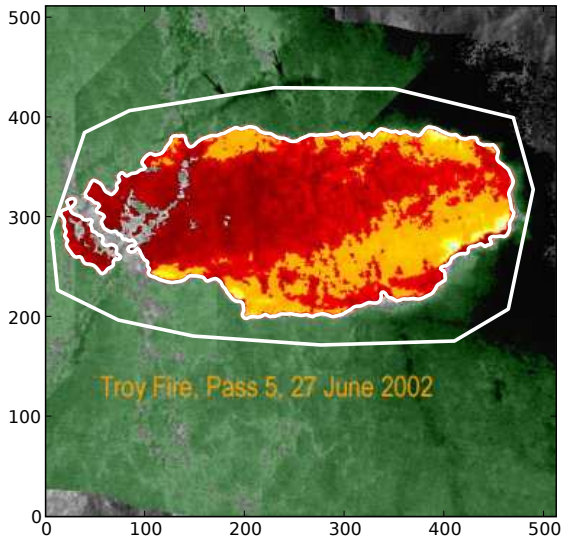



Figure 5: Image segmented: initial and final fronts are in white.

## 2.4 Image Segmentation and Multivac Configuration Files

In Multivac package, program `image` is used to perform image segmentation. It uses a configuration file like `image.cfg`. This configuration file contains the paths to results and base image, and it contains the description of the simulation domain. In order to load the image and the results pointed into the configuration file, one may use `load_segmented`:

```
In [19]: xrange, yrange, image, front = load_segmented("image.cfg")
```

One-dimensional arrays `xrange` and `yrange` store coordinates of image pixels. Two-dimensional array `image` stores the segmented image, and `front` includes all front saved. The results may be displayed following the steps of section 2.3, or with function `display_all`:

```
In [20]: display_all(xrange, yrange, image, front, 'w-')
```

Figure 6 shows the output.

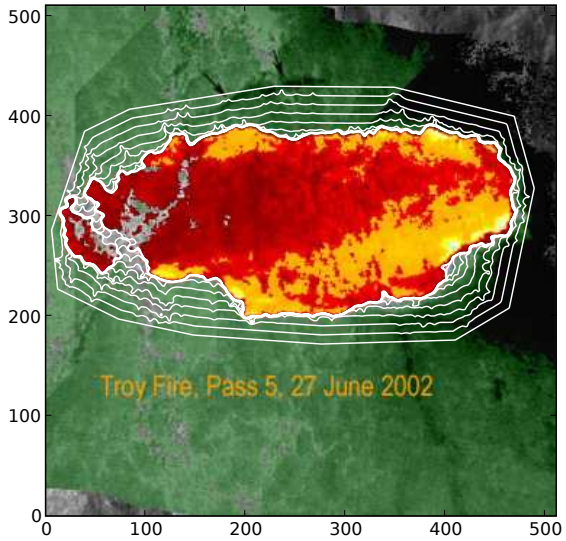



Figure 6: Image segmented, with all fronts, as displayed by `display_all`.

# 3 Graphical User Interface

## 3.1 Launching the GUI

One needs Multivac version 1.10 or higher to run the graphical user interface. In Multivac root directory, compile program `image` and launch `./gui.py`.

Notice that `gui.py` imports MultivacDisplay. Python will search for it in the local directory and, then, in the paths in the shell variable `$PYTHONPATH` (just like variable `$PATH` is used by Bash or any similar shell). Therefore, to use the GUI, it is necessary to put it in the Multivac root directory or in `$PYTHONPATH`.

## 3.2 Description of the GUI

Once launched, the GUI returns a window shown in Figure 7.

Open the sample image `star-red.png` that is in Multivac root directory: in menu "File", choose "Open image", or simply press `ctrl+o`. This should load and display the image in the main window.

Then you need to preprocess the image before Multivac segments it. Press button "Preprocessing" to perform this first step. It may take some time. If the file has already been preprocessed, there is no need to launch the preprocessing: you may put the path to the preprocessed file in the line edit.

Next you need to draw the initial front around the star. Use mouse left button to add vertices and mouse right click to close the curve. You may clean what you have drawn with a middle click. Figure 8 shows a possible initial front.

Finally press "Save all & proceed" to have the image segmented. A result is shown in Figure 9. Buttons "Save" and "Save configuration" may be used instead in order to save the initial front and to save the configuration (for program `image`) respectively. This is useful for anyone who wants to work outside the GUI after these steps.

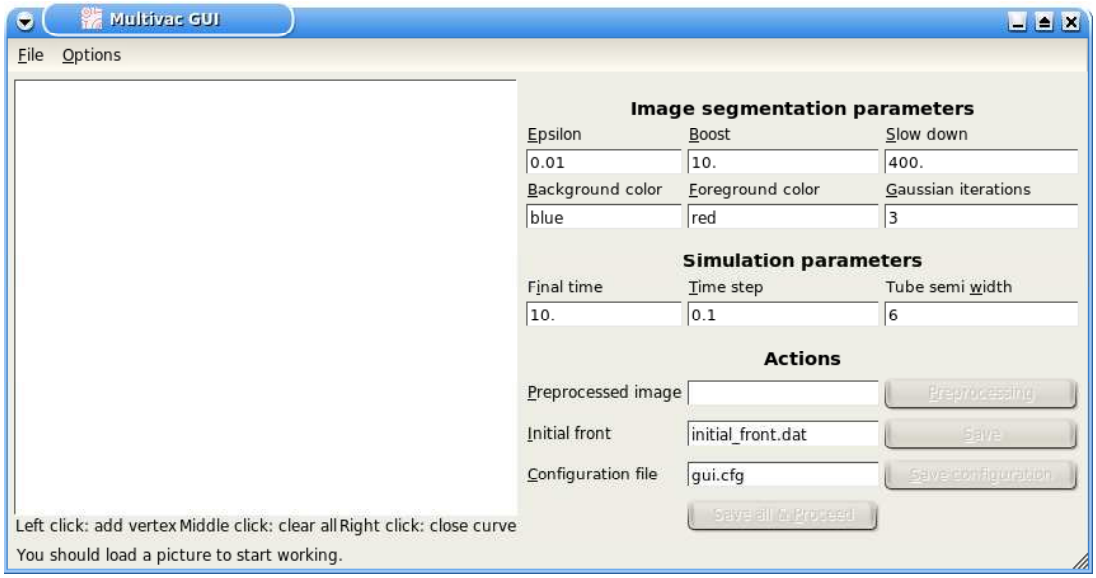



Figure 7: Graphical user interface as it appears at start.

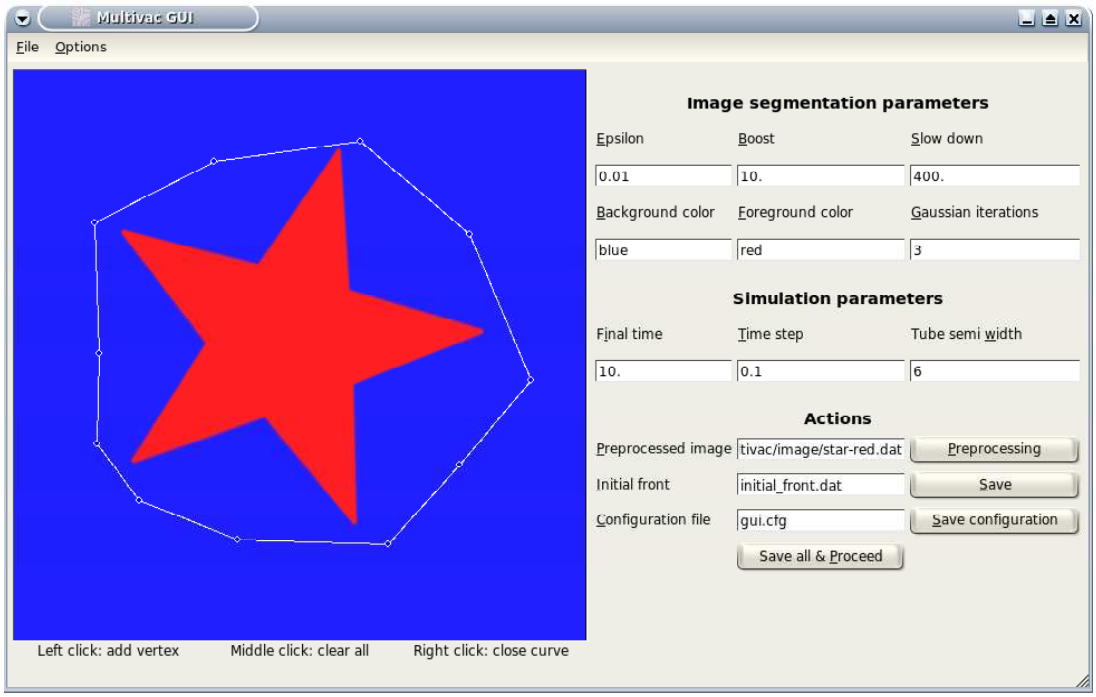



Figure 8: Graphical user interface after the initial front as been drawn.

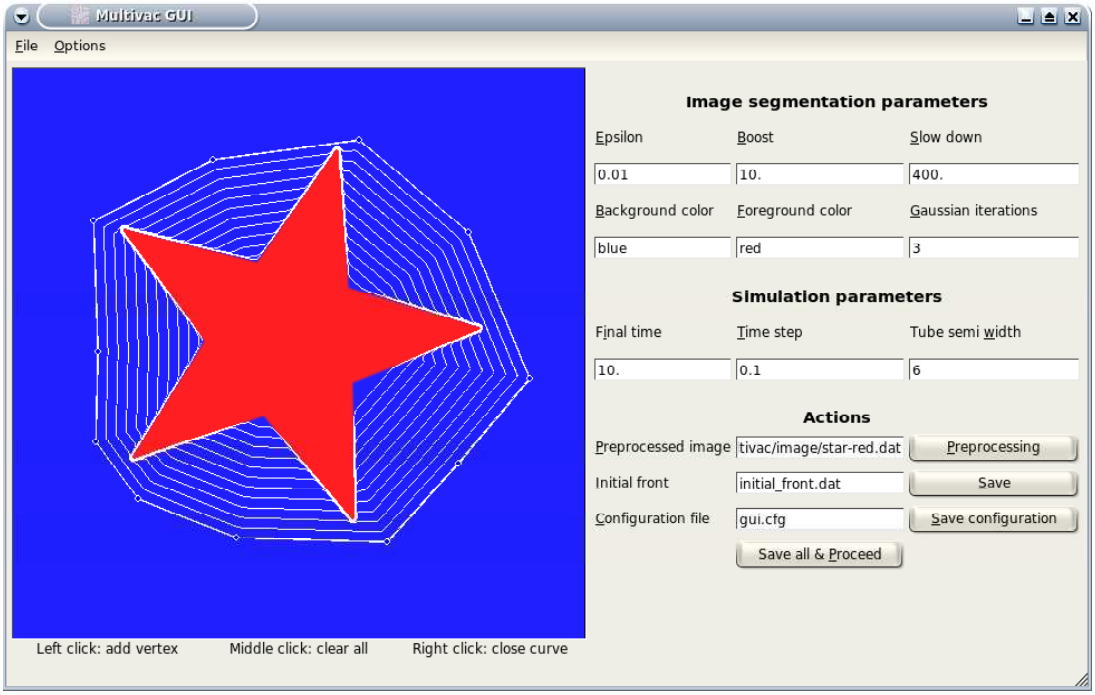



Figure 9: Graphical user interface after the star contours are partially determined. One may increase the final time to segment the whole star.

It is possible to clean the results (mouse middle click) and to start a new segmentation, possibly with new parameters. Several parameters appear in the GUI. They refer to the parameters for image preprocessing and for image segmentation. One should study the underlying algorithms to grasp their meanings.