LithoGraphX Documentation

Release 1.2.0

Barbier de Reuille, Pierre <pierre@barbierdereuille.net>

August 03, 2016

Contents

LithoGraphX is a software to visualize, process and analyse 3D images and meshes. This documentation is organized into a reference manual describing the user interface in details and a collection of tutorials describing how to perform usual tasks.

The main website of LithoGraphX is: http://www.lithographx.org

LithoGraphX is a fork of the MorphoGraphX project.

Hint: If you have never used LithoGraphX, I recommend you start by reading the definitions and the description of the main interface. Then, you should probably read tutorials close to what you want to do.

Installation

1.1 Hardware Requirements and Recommendations

Volumetric rendering and 3D image processing are very demanding tasks both for the CPU and the GPU. For this reason, it is highly recommended to opt for high-end desktop computer. Also, both because LithoGraphX uses CUDA and because their video drivers tend to be more stable we highly recommend using NVidia GPUs.

1.1.1 Recommended configuration

As of July 2016, here are recommended configurations which should offer a good price/performance ratio:

Desktop Computer

- CPU: Intel Core i7 6700K with 32 GB of RAM

- GPU: NVidia GeForce GTX 970 with 4GB of dedicated memory

Laptop Computer

- CPU: Intel Core i7-4720HQ with 16 GB of RAM

- GPU: NVidia GeForce GTK 960M with 2GB of dedicated memory

1.1.2 Minimum configuration

Desktop Computer

- CPU: Intel Core i7 with 8GB to 16GB of RAM (depending on the size of your images)

- GPU: any NVidia or AMD GPU less than 5 years old.

Laptop Computer

- CPU: Intel Core i7 with 8GB to 16GB of RAM (depending on the size of your images)

- GPU: high-end NVidia GPU of Intel HD Graphics 4600 or more recent with the latest video drivers (tested with drivers Build 10.18.15.4248). We do not recommend AMD GPUs due to serious issues with their video drivers for laptops.

1.2 Install from binaries

1.2.1 Microsoft Windows

Quick Setup

1. Install LithoGraphX for Windows: http://updates.lithographx.org (the default version requires an NVidia graphics cards, otherwise download the NOCUDA version)

2. Install Anaconda Python 2.7 64 bits: https://www.continuum.io/downloads

Full Setup

If you don't want to install Anaconda or would rather know the full list of Python modules used in LithoGraphX, please read on.

First, to run LithoGraphX, you will also need to install the 64 bits version of Python 2.7 for Windows. There are many sources from which you can download Python for windows. We recommend using Anaconda (https://www.continuum.io/downloads), as it will include by default a large number of libraries for scientific computing and machine learning. However, you can also download the official python binaries from there: Python 2.7.11 64 bits.

If you install the official python binaries, you may need to install Numpy, SciPy and scikit-learn (all of which are provided by Anaconda) as they are used by some macros. First, install Numpy and Scipy by downloading the corresponding wheel files from this website: http://www.lfd.uci.edu/~gohlke/pythonlibs/. Once the files downloaded, you can use `pip` to install them. For example, at the time this is written, you will download the files `numpy-1.10.4+mkl-cp27-cp27m-win_amd64.whl` for numpy, `scipy-0.17.0-cp27-none-win_amd64.whl` for scipy and `pandas-0.17.1-cp27-none-win_amd64.whl` for pandas. To install them, open a command prompt and type:

```
pip install numpy-1.10.4+mkl-cp27-cp27m-win_amd64.whl scipy-0.17.0-cp27-none-win_amd64.whl pandas-0.1
```

`pip` can also be used to download and install scikit-learn:

```
pip install scikit-learn
```

You will find all binary installers in the LithoGraphX update site: http://updates.lithographx.org. LithoGraphX for windows is currently compiled in two version:

- a version requiring CUDA, this is the default version.

- a version that doesn't use CUDA, named `-NOCUDA`.

Download the version you want (probably the latest) and execute it to install LithoGraphX.

Note: Since the version 1.1.1, LithoGraphX for Windows will check if a new version exist, and will offer to update your installation. Automatic checking for a new version can be tweaked/deactivated from the settings dialog box.

Hint: If you have issues, check the *FAQ on installation*.

1.2.2 Linux - Ubuntu

The simplest way to install LithoGraphX and to keep it up to date is to use the launchpad PPA. To use it you have two options:

1. From the Ubuntu Software Center, select the menu Edit->Software Sources. In the `Other software` tab, click the `Add...` button and enter `ppa:pierre-barbierdereuille/ppa` in the `APT line`. For more details look at the answer of this question:

   http://askubuntu.com/questions/4983/what-are-ppas-and-how-do-i-use-them

   The PPA offers 4 packages. The two main ones are `lithographx` and `lithographx-nocuda`. As the name suggests, `lithographx-nocuda` simply doesn't use CUDA. These two packages contain the latest tested version of LithoGraphX. The two other packages are `lithographx-daily` and `lithographx-nocuda-daily`. As their name suggests, these two packages are compiled daily with whatever the latest version of the source code is. As such, they are not recommended for non-developers.

2. You can use the command line and type from a terminal:

```
$ sudo add-apt-repository ppa:pierre-barbierdereuille/ppa
$ sudo apt-get update
$ sudo apt-get install lithographx
```

Hint: If you have issues, check the *FAQ on installation*.

Important: After installing LithoGraphX on Linux, you need to *configure the Alt key*.

1.2.3 Linux - other

If you don't use an Ubuntu-derived distribution but still a debian-derived one, you can obtain the source package and build it yourself:

1. Enable source downloading for the PPA:

```
$ sudo add-apt-repository -s ppa:pierre-barbierdereuille/ppa
```

2. Create a folder where you are going to build the package. From this folder, type:

```
$ apt-get source -b lithographx
```

   If you have problems building, or want to adjust the dependencies for your system, you can also first get the source code:

```
$ apt-get source lithographx
```

   Make the necessary adjustments to the package, and build it. For the version 1.1.0, you can build the package with:

```
$ cd lithographx-1.1.0
$ dpkg-buildpackage -us -uc -nc
```

   The debian package will be created in the folder containing `lithographx-1.1.0`.

For non-debian based systems, you will need to install from source. If you need help doing so, do not hesitate to contact us.

Important: After installing LithoGraphX on Linux, you need to *configure the Alt key*.

1.3 Install from source

First, you need to retrieve the source code, which is hosted by BitBucket: https://bitbucket.org/PierreBdR/lithographx

Then, you need to follow the instructions for your operating system.

1.3.1 Requirements

To compile LithoGraphX you will need the following programs installed (more recent versions should work too):

- g++ 4.8: https://gcc.gnu.org/
- cmake 2.8.11: http://www.cmake.org
- Qt 5.2 or greater with OpenGL module: http://www.qt.io
- CImg: http://cimg.sourceforge.net/
- GNU Scientific Library 1.16: http://www.gnu.org/software/gsl/gsl.html
- LibTIFF v4 from http://libtiff.maptools.org/
- python 2.7 or 3.4+: http://www.python.org
- sip 4.15: http://www.riverbankcomputing.com/software/sip/download
- PyQt 5.2: http://www.riverbankcomputing.com/software/pyqt/download5
- cuda: https://developer.nvidia.com/cuda-zone

To run some of the macros, you will need:

- NumPy 1.8: http://www.numpy.org
- SciPy 0.13: http://www.scipy.org
- Pandas 0.17: http://pandas.pydata.org
- scikit-learn 0.16: http://scikit-learn.org

To generate the documentation, you will need:

- doxygen: http://www.stack.nl/~dimitri/doxygen/
- sphinx: http://sphinx-doc.org/

1.3.2 Installation on Linux

If using Ubuntu, or probably any debian-derived system, you will find a list of packages to install in the file called ubuntu-packages.txt at the root of the source tree.

In the list of dependencies, the packages `nvidia-cuda-toolkit` may be replaced with any package from NVidia (e.g. `cuda-7-0`), or simply by `libthrust-dev` if you want to compile without CUDA.

For other distribution, you will need to find equivalent packages.

Configuration

Create a folder in which the application will be built. Then, using `cmake-gui`, select the source and build directories. Make sure `CMAKE_BUILD_TYPE` is set to `Release`, and select the modules you want to build by checking the variables starting with `BUILD_`. When configuring, more options may appear (in red), in this case, just configure again, until you have no error and no red line. When that is done, click the `Generate` button.

Compilation

Open a shell in the build folder and simply use `make`. To speed up compilation, you can choose to use parallel compilation with the `-j` options. For examplem to use 8 compilation processes in parallel, type: `make -j8`.

1.3.3 Installation on Windows

Configuring your development environment for Windows is a complex task. We are using gcc for the compilation (to be able to use the GNU Scientific Library), but at the same time, python and CUDA require the use of Visual Studio (at least by default). For that reason, windows has its own page:

Installing LithoGraphX from Source on Microsoft Windows

There are two ways of compiling for Windows: the recommended way using *MinGW64* or using *Visual Studio*.

Compilation with MinGW64 – The Easy Way

The easier way to compile with MinGW64 is to install it via MSys2: http://sourceforge.net/projects/msys2/

Here is a step by step method to install all the libraries you need for LithoGraphX.

1. (Cuda version only) Install Visual Studio 2013, for example the community edition which is available for free on http://www.visualstudio.com. From this version, you only need the C++ component.

2. (Cuda version only) Install the CUDA drivers. **Note:** These must be installed after Visual Studio, so CUDA find it and install all the necessary components.

3. Install MSys2 using the graphical installer

4. Open the MSys2 Win64 shell using the shortcut installed in the start menu.

5. Upgrade MSys2 using `pacman`, the package manager of MSys2:

```
$ pacman -Su
```

6. Install the packages we need for LithoGraphX:

```
$ pacman -S mingw-w64-x86_64-libtiff mingw-w64-x86_64-qt5 mingw-w64-x86_64-python2-pyqt5 mingw-w
```

7. (Optional). Install VTK:

```
$ pacman -S mingw-w64-x86_64-vtk mingw-w64-x86_64-opencv
```

We also need to install OpenCV as it is a dependence of VTK. However, as far as the version 20150512 of MSys2 is concerned, OpenCV is not installed automatically with VTK.

8. (Optional). Install ITK. Sadly, the version of ITK that comes with MSys2 doesn't include the auto-seeded morphological watershed, central in so many protocols of LithoGraphX. So we need to compile it ourselve:

   (a) Download ITK from http://www.itk.org

   (b) Prepare your directory structure. For simplicity, I will assume everything linked to ITK will be in the same folder. This folder will contain three sub-folder:

   > ITK_Source
   >
   > > Contains the source code of ITK
   >
   > ITK_Build
   >
   > > Will contain the build files of ITK
   >
   > ITK
   >
   > > Will contain the installed version of ITK

   (c) Make sure the three directories exist, and `ITK_Source` contains the source code.

   (d) Launch a MSys2 Win64 shell, using the shortcut in the start menu.

   (e) Go to the ITK folder. Careful: MSys2 has an odd syntax. For example, if your folder is `C:\Users\Me\ITK`, then you need to type:

   ```
   $ cd /C/Users/Me/ITK
   ```

   (f) Configure ITK:

   ```
   $ cd ITK_Build
   $ cmake -G "MinGW Makefiles" -DCMAKE_CXX_FLAGS="-std=c++11" -DCMAKE_BUILD_TYPE=Release -DBUI
   ```

   (g) Then, compile and install ITK:

   ```
   $ make -j4 install
   ```

9. (Optional). Install the GNU Scientific Library:

   ```
   $ pacman -S -s mingw-w64-x86_64-gsl
   ```

10. (NoCuda version only) Download and extract the thrust library https://thrust.github.io. As this library only contains header files, there is nothing to compile. Simply mark down where it is installed.

11. (Cuda version only) Create batch scripts to have both MinGW64 and Visual Studio compilers available. For the default installation of Visual Studio and MinGW64, the script should contain:

    ```
    call "C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.bat" amd64
    @echo Visual Studio installed in current shell
    @set MINGW64=C:\msys64\mingw64
    @set PATH=%MINGW64%\bin;%PATH%
    @echo MINGW64 installed in current shell
    ```

    You might also want to create a shortcut to start the shell. Right-click somewhere (for example the desktop) and select `New->Shortcut`. In the dialog box that appear, for location type:

    ```
    %comspec% /k ""C:\Path\to\script.bat""
    ```

    For the following, use the shortcut to open a terminal, and go to the LithoGraphX source tree:

    ```
    $ cd C:\Users\Me\LithoGraphX
    ```

12. (NoCuda version only). Open an msys Win64 shell from the start menu, and go to the LithoGraphX source tree:

```
$ cd /C/Users/Me/LithoGraphX
```

13. Create a `Build` folder in the LithoGraphX source tree:

```
$ mkdir Build
$ cd Build
```

14. Configure LithoGraphX:

```
$ cmake-gui ..
```

   (a) (Cuda version) Select `MinGW Makefiles`

   (b) (NoCuda version) Select `MSys Makefiles`

   (c) On the right of the search box, check the boxes `Grouped` and `Advanced`.

   (d) Press the `Configure` button and check for errors until there are none left. Typically, you will need to inform the following variables:

   PYTHON_EXECUTABLE

       CMake will find the program `python.exe`, but really you want `python2.exe` in the same folder

   PYTHON_LIBRARY

       If you already have python installed, CMake may find it instead of the python for MinGW64. If you left the default installation folder of MSys2, the correct path for the library is `C:/msys64/mingw64/lib/libpython2.7.dll.a`

   SIP_EXECUTABLE

       This is the path to the `sip.exe` program. It should be at `C:/msys64/mingw64/bin/sip.exe`.

   SIP_PYQT5_CORE_PATH

       This is the path to the Core module of PyQt5. This should be `C:/msys64/mingw64/share/sip/Py2-Qt5/QtCore`

   SIP_PYQT5_PATH

       This is the path to all the modules of PyQt5. This should be: `C:/msys64/mingw64/share/sip/Py2-Qt5`

   THRUST_BACKEND_CUDA / THRUST_BACKEND_OMP

       Check one or the other depending on whether you want to compile with CUDA or not. If not, you can delete the whole CUDA group.

   CMAKE_BUILD_TYPE

       Set this one to `Release` to compile with all the optimizations.

   CMAKE_INSTALL_PREFIX

       If you don't want to install LithoGraphX into the `Program Files` folder, change the value. Note that to install in `Program Files`, you need to set the proper permissions, unless you use the installer.

   BUILD_ITK

       If you have build ITK, check this box. The last build of ITK should be found automatically.

BUILD_VTK

If you have installed VTK, check this box. Again, CMake should find where it is automatically.

(e) When you are done configuring, press the `Generate` button. If there is no error, you are done with configuring LithoGraphX.

15. Now, you need to compile LithoGraphX. If you use the MSys command line, you need to type:

```
$ make -j8
```

Otherwise, you need to type:

```
$ mingw32-make
```

16. Once compiled, you can either directly install LithoGraphX, or create a package you can re-distribute. We recommend the creation of a package. For this, from the command line, type:

```
$ cpack -G NSIS64
```

Compilation with MinGW64 – The Hard Way

This is another way to get MinGW64 with all the libraries. It avoids compiling MinGW64 and Qt, but almost all the other dependencies are compiled.

Because CUDA requires Visual C++ under windows, and because we are using MinGW64 for the rest of the compilation, you have to be careful when installing your development environment. Here, I will describe how to install it for CUDA 7.0:

1. Install Visual Studio 2013, for example the community edition which is available for free on http://www.visualstudio.com. From this version, you only need the C++ component.

2. Install the CUDA drivers. **Note:** These must be installed after Visual Studio, so CUDA find it and install all the necessary components.

3. Install python. We highly recommend using Anaconda. Once installed, you will need to install the `libpython` conda package.

4. Download the source code of SIP and PyQt5 from http://www.riverbankcomputing.com

5. Download the version of Qt5 corresponding to the version of PyQt5 you downloaded from http://sourceforge.net/projects/qt64ng **Note:** This version of Qt5 also contains python 2.7. However, to make it work you need to go into `mingw64\opt\bin` and copy the file `libpython2.7.dll` to `mingw64\libs\python27.dll` (the folder doesn't exist)

6. Download MSys from the MinGW64 website here: http://sourceforge.net/projects/mingw-w64/files/External%20binary%20packages%20%28Win64%20hosted%29/

7. To make your life simpler, create two scripts that will add the path to MinGW64 to your PATH environment variable: one for the command shell, and one for UNIX shell. Assuming you installed MinGW64 in `C:\MinGW64`, the first script should be called `mingw64.bat` and have for content:

```
@set MINGW64=C:\MinGW64
@set PATH=%MINGW64%\bin;%MINGW64%\opt\bin;%MINGW64%\opt\libs;%PATH%
```

The second script should be called `mingw64.sh` and have for content:

```
export PATH=/c/MinGW64/bin:$PATH
```

This way, when you open a terminal, if this is a windows command, you can type:

```
$ call mingw64.bat
```

and have MinGW64 available, or for MSys shell:

```
$ source mingw64.sh
```

8. **Compilation of SIP**

   (a) Start a windows command and setup your environment variables for MinGW64:

   ```
   $ call mingw64.bat
   ```

   (b) Configure sip:

   ```
   $ python configure.py -p win32-g++
   ```

   (c) Compile and install:

   ```
   $ mingw32-make
   $ mingw32-make install
   ```

   (d) Copy the file `sip.exe` in `%MINGW64%\opt` to `%MINGW64%\opt\bin`

Compilation of PyQt5

   1. Start a windows command and setup your environment variables for MinGW64 (you can re-use the one open for SIP)

   2. Configure PyQt5:

   ```
   $ python configure.py --spec=win32-g++
   ```

   3. Compile and install:

   ```
   $ mingw32-make
   $ mingw32-make install
   ```

   4. Copy the files in `%MINGW64%\opt` to `%MINGW64%\opt\bin`

1. Download the source code of libtiff5 from the GNUWin32 project: http://gnuwin32.sourceforge.net/packages/tiff.htm

   (a) Launch the MSys shell

   (b) Setup the environment variables for MinGW64

   (c) Create a build directory and change to it (optional)

   (d) Run the configure script, specifying a prefix

   (e) Compile and install

2. (Optional). Download and compile ITK at: http://www.itk.org

   (a) Prepare your directory structure. For simplicity, I will assume everything linked to ITK will be in the same folder. This folder will contain three sub-folder:

   ITK_Source

   > Contains the source code of ITK

   ITK_Build

   > Will contain the build files of ITK

ITK

Will contain the installed version of ITK

(b) Make sure the three directories exist, and `ITK_Source` contains the source code.

(c) Launch a windows command and setup your environment variables for MinGW64

(d) From the main directory, configure ITK:

```
$ cd ITK_Build
$ cmake -G "MinGW Makefiles" -DCMAKE_CXX_FLAGS="-std=c++11" -DCMAKE_BUILD_TYPE=Release -DBUI
```

(e) Then, compile and install ITK:

```
$ mingw32-make install
```

3. (Optional). Download and compile VTK at: http://www.vtk.org

(a) Prepare your directory structure. For simplicity, I will assume everything linked to VTK will be in the same folder. This folder will contain three sub-folder:

VTK_Source

Contains the source code of VTK

VTK_Build

Will contain the build files of VTK

VTK

Will contain the installed version of VTK

(b) Create a build directory, for example call `VTK_build`

(c) Launch a windows command and setup your environment variables for MinGW64

(d) From the main directory, configure VTK:

```
$ cd VTK_Build
$ cmake -G "MinGW Makefiles" -DCMAKE_CXX_FLAGS="-std=c++11" -DCMAKE_BUILD_TYPE=Release -DCMA
```

(e) Then, compile and install VTK:

```
$ mingw32-make install
```

4. (Optional, for extra modules only) Download the GNU scientific library from their website.

(a) Prepare your directory structure. For simplicity, I will assume everything linked to the GSL will be in the same folder. This folder will contain two sub-folder:

GSL_Source

Contains the source code of the GSL

GSL

Will contain the installed version of the GSL

(b) Launch the MSys shell

(c) Setup the environment variables for MinGW64

(d) Create a build directory and change to it (optional)

(e) Run the configure script, specifying a prefix:

```
$ cd GSL_Source
$ configure --prefix=../GSL
```

(f) Compile and install:

```
$ make && make install
```

5. Configure LithoGraphX for compilation

   (a) Create a build directory

   (b) Start a x64 Native Tools Command Prompt from the Visual Studio tools.

   (c) Setup the MinGW64 environment variables:

```
$ call mingw64.bat
```

   (d) Change the current folder to the build directory you created

   (e) Run cmake-gui, specifying the path to the source. For example, if you created your build folder as a sub-folder in the source tree:

```
$ cmake-gui ..
```

   (f) Select the `MinGW Makefiles` generator

   (g) Check the `Grouped` check box to simplify the interface. This will put together all the options starting with the same work.

   (h) Press the `Configure` button and check errors until there are none left. Typically you will need to inform the following variables have (in order in which they will appear):

   CUDA_CL_VERSION

   > You need to set this to your version of Visual Studio, so here 2013.

   TIFF_INCLUDE_DIR

   > This should be the `include` of the installation folder for libtiff.

   TIFF_LIBRARY

   > This should be the file `libtiff.a` in the `lib` folder of the installation of libtiff.

   Eigen_SIGNATURE

   > You need to select the file `signature_of_eigen3_matrix_library` in the Eigen folder.

   SIP_EXECUTABLE

   > If this is not found by default, this is because you didn't install sip in the default python location. It should be in the same folder as the python you used to compile it. If this is the case, you should activate the advanced options and make sure you select the python interpreter SIP is compiled for.

   SIP_INCLUDE_PATH

   > This is the `include` folder, wherever the sip executable is

   SIP_PYQT5_PATH

   > This should be the `sip\PyQt5` folder, still where the sip executable is.

   PYTHON_LIBRARY

To access this variable, you need to show the advance options. By default, this will look for `python27.lib`. If you use Anaconda, you need to install `libpython` and select the file `libpython27.a` instead.

BUILD_EXTRA_INCLUDES

If you didn't install CImg in a standard path, chances are, you will need to add its path in this variable, for example if it is not found. This shouldn't be a problem if you copy CImg.h in the `src` folder of LithoGraphX.

CMAKE_BUILD_TYPE

Set this to `Release` for maximal performance.

CMAKE_INSTALL_PREFIX

Choose the folder where LithoGraphX will be installed. If this is in `Program Files`, make sure the folder already exists and you can write in it, as normal programs cannot create a folder there.

BUILD_ITK

If you have built ITK, check this box

BUILD_VTK

If you have built VTK, check this box

6. Without closing the command line, start the compilation:

```
$ mingw32-make
```

7. Then, install:

```
$ mingw32-make install
```

You can contact the development team to get a zip file with all the versions we are using currently.

Compilation with Visual Studio 2013

Note the following process should also work with other versions of Visual Studio. However, I haven't tested it, so it is not granted.

1. Install Visual Studio 2013, for example the community edition which is available for free on http://www.visualstudio.com. From this version, you only need the C++ component.

2. Install the CUDA drivers. **Note:** These must be installed after Visual Studio, so CUDA find it and install all the necessary components.

3. Download the source code of SIP and PyQt5 from http://www.riverbankcomputing.com

4. Download the version of Qt5 corresponding to the version of PyQt5 you downloaded.

5. Download Python 2.7 or 3.4. If you are an academic, I recommend using the Anaconda distribution.

6. **Compilation of SIP**

   (a) Start a x64 Native Tools Command Prompt from the Visual Studio tools.

   (b) Configure sip. At this time, only Visual Studio 2010 is supported, but it seems to work for Visual Studio 2013 too:

```
$ python configure.py -p win32-msvc2010
```

(c) Compile and install:

```
$ nmake
$ nmake install
```

7. **Compilation of PyQt5**

   (a) Start the same command prompt used for SIP (you can re-use the one open for SIP)

   (b) Configure PyQt5:

```
$ python configure.py --spec=win32-msvc2013
```

   (c) Compile and install:

```
$ nmake
$ nmake install
```

8. Download the source code of libtiff5 from the GNUWin32 project:
   http://gnuwin32.sourceforge.net/packages/tiff.htm

   (a) Launch the x64 Native Tools Command Prompt

   (b) Compile:

```
$ nmake /f Makefile.vc
```

   (c) The programs may fail to compile, but the library should be fine. It will be in the `libtiff` folder.

9. Download CImg from http://cimg.sourceforge.net/. Just uncompress it somewhere: this is only a header file, so there is no compilation necessary. Also, copy the header file of CImg in the `src` folder of LithoGraphX.

10. (Optional). Download and compile ITK at: http://www.itk.org

   (a) Prepare your directory structure. For simplicity, I will assume everything linked to ITK will be in the same folder. This folder will contain three sub-folder:

```
ITK_Source
```

        Contains the source code of ITK

```
ITK_Build
```

        Will contain the build files of ITK

```
ITK
```

        Will contain the installed version of ITK

   (b) Make sure the three directories exist, and `ITK_Source` contains the source code.

   (c) Launch a VS2013 x64 Native Tools Command Prompt

   (d) From the main directory, configure ITK:

```
$ cd ITK_Build
$ cmake -G "NMake Makefiles" -DCMAKE_BUILD_TYPE=Release -DBUILD_TESTING=OFF -DBUILD_EXAMPLES
```

   (e) Then, compile and install ITK:

```
$ nmake install
```

11. (Optional). Download and compile VTK at: http://www.vtk.org

    (a) Prepare your directory structure. For simplicity, I will assume everything linked to VTK will be in the same folder. This folder will contain three sub-folder:

        `VTK_Source`

            Contains the source code of VTK

        `VTK_Build`

            Will contain the build files of VTK

        `VTK`

            Will contain the installed version of VTK

    (b) Create a build directory, for example call `VTK_build`

    (c) Launch a VS2013 x64 Native Tools Command Prompt

    (d) From the main directory, configure VTK:

```
$ cd VTK_Build
$ cmake -G "NMake Makefiles" -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=..\VTK ..\VTK
```

    (e) Then, compile and install VTK:

```
$ nmake install
```

12. Create a `build` folder, either in or outside of the LithoGraphX folder

13. Run cmake, and select correct the source code and build binary folders

    (a) You can use either the `Visual Studio 2013 Win64` or the `NMake Makefiles` generator.

    (b) Check the `Grouped` check box to simplify the interface. This will put together all the options starting with the same work.

    (c) Press the `Configure` button and check errors until there are none left. Typically you will need to inform the following variables have (in order in which they will appear):

        CUDA_CL_VERSION

            You need to set this to your version of Visual Studio, so here 2013.

        CUDA_HOST_COMPILER

            You need to put here the path to the cl program. To get this, from the command line you have opened earlier, type:

```
$ where cl
```

            And copy the path writen.

        TIFF_INCLUDE_DIR

            This should be the `libtiff` sub-folder where you compiled libtiff in a previous step

        TIFF_LIBRARY

            This should be the file `libtiff.lib` in the same folder as for TIFF_INCLUDE_DIR

        Eigen_SIGNATURE

            You need to select the file `signature_of_eigen3_matrix_library` in the Eigen folder.

        SIP_EXECUTABLE

> If this is not found by default, this is because you didn't install sip in the default python location. It should be in the same folder as the python you used to compile it. If this is the case, you should activate the advanced options and make sure you select the python interpreter SIP is compiled for.
>
> SIP_INCLUDE_PATH
>
> > This is the `include` folder, wherever the sip executable is
>
> SIP_PYQT5_PATH
>
> > This should be the `sip\PyQt5` folder, still where the sip executable is.
>
> BUILD_EXTRA_INCLUDES
>
> > If you didn't install CImg in a standard path, chances are, you will need to add its path in this variable, for example if it is not found. This shouldn't be a problem if you copy CImg.h in the `src` folder of LithoGraphX.
>
> CMAKE_INSTALL_PREFIX
>
> > Choose the folder where LithoGraphX will be installed. If this is in `Program Files`, make sure the folder already exists and you can write in it, as normal programs cannot create a folder there.
>
> (d) Press the `Generate` button

14. If you used the `NMake Makefile` generator, open a x64 Native Tools Command Prompt, and compile LithoGraphX:

```
$ nmake install
```

Otherwise, open the file `ALL_BUILD.vcxproj`, select the `Release` compilation (the default is `Debug`), click on the `INSTALL` solution and compile with the `Build INSTALL solution` entry from the `Build` menu. **Note:** The entry will only be available if the `INSTALL` solution is the one currently selected on the left pane.

1.4 Configuring the Operating System

1.4.1 Configuring the *Alt* key under Linux

Most window manager are configured to use the `Alt` key in combination with the mouse to move or resize windows. On the other hand, LithoGraphX uses the `Alt` key to interact with the mesh and the stack. So to use LithoGraphX, the window manager needs to be configured to use another key, such as the `Super` or `Meta` key (i.e. the window key on your keyboard). Here is how to do it on some common window manager:

Gnome 3 TODO

Cinnamon In the system settings, find the `Windows` settings. There, change the mouse modifier key from `Alt` to `Super`.

KDE In the system settings, open the `Window behaviour` settings. Then, on the `Window Actions` tab, change the modifier key to `Meta`.

XFCE 4 In the Settings, select `Window Manager Tweaks`. From there, in the `Accessibility` tab, change the `Key used to grab and move windows` from `Alt` to `Super`.

Principles and user Interface

LithoGraphX is an open source platform for the visualization and processing of 3D image data. It can visualise and manipulate 3D image stacks and triangulated curved surfaces (2½D). Surfaces can be extracted from 3D images and both can interact in various ways.

Before introducing the user interface there are a few useful definitions that come over and over in LithoGraphX.

2.1 Definitions

Stack A stack is really a biological sample. Each stack has a unique size, location and resolution. It contains 2 images (stores) and a mesh. The two stores of a stack are the main and the work store. The main store is where the data is loaded by default. When the data is processed, the result is always put in the work store, using the main store as a checkpoint. They can also be used to load two different channels of a same image.

Store A store is a 16 bits, single channel, 3D image. Each store has its own rendering properties (color, opacity, brightness), but they share location, resolution and size. The voxel values can be interpreter either as signal intensity or labels.

Mesh A mesh is a triangulated surface. A mesh can be loaded from a file, or extracted from a 3D image. It is worth noting the surface doesn't have to be continuous and can therefore represent many distinct objects.

Process——- All non-interactive operations in LithoGraphX are performed by processes. They are grouped into three categories: Stack, Mesh and Global. Stack and processes cannot modify meshes, Mesh processes cannot modify Stack or Stores and Global processes can modify anything. Processes can be used in Python scripts or macros to automate complex pipelines. Processes are refered as `Type.Name`. For example, the process `Stack.Save` is a stack process used to save the current stack.

Active Stack At any point in time, a single store is marked as active. The active store is the one processed by default. If a process works on meshes, the active mesh is the one associated with the active store. Before launching a process, you should always check which is the active store: it is indicated on the bottom right of the main window.

2.2 User Interface

The main window (Figure 2.1) of LithoGraphX is organised around a central interaction area, black when you stack the application. This is also where you want to drop files to load them in their default place.

The volume and mesh toolbars offer tools to interact with either the volume or the mesh. This tools are always activated by pressing the Alt key and clicking on the interaction area. You will find a complete description of the tools in *Interaction toolbars*.
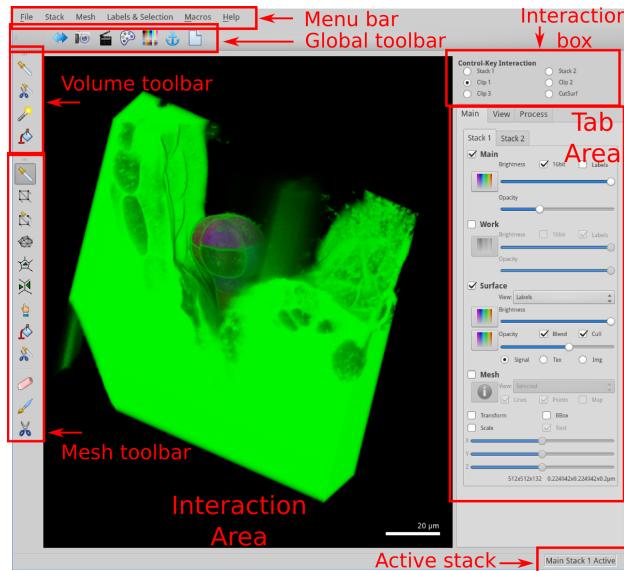
Fig. 2.1: User Interface

2.2.1 Interaction area

This is the most important area of LithoGraphX. This is where your objects are displayed and where you can interact with them. The commands used for this area are available either by pressing the H key when hovering above the interaction area, or by selecting "Mouse and Keyboard" in the help menu.
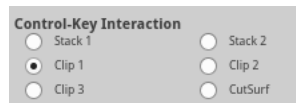
2.2.2 Interaction box

Fig. 2.2: Interaction box

The object selected in the interaction box (Figure 2.2) can be moved around by presing the **control** key and using the same mouse movement as for moving the camera.

2.2.3 Global toolbar

Fig. 2.3: Global toolbar

See Figure 2.3.

Current label The most important function of the global toolbar is to visualize and reset the current label. The left-most button (transparent in the image above), indicate which is the current label. You can see its color on the tool and the tooltip will give you its exact number. Cliking on it will reset the label to 0 (e.g. no label selected).

Swap surfaces This tool is used when co-segmenting meshes and is described in the appropriate protocol.

Save screenshot As the name suggest, this tool brings the screenshot interface to capture the interaction area in an image.

Record video This allows to take a screenshot every time the image changes. This can be handy to record a movie.

Color palette This tool opens a dialog box used to change the colors used in the interaction area: background, mesh lines, text, ...

Edit labels This tool opens a dialog box used to change the number of colors used when representing labeled stacks or meshes. It also allows to change which colors are used for which label.

Reset view This is a panic button when you loose your objects. It will reset the view, including the positions of the stacks, the clipping planes and the cutting surfaces you may have moved.

Edit project file This tool opens a simple text editor from which you can edit the project file directly. This is for advanced users only!

2.2.4 Interaction toolbars
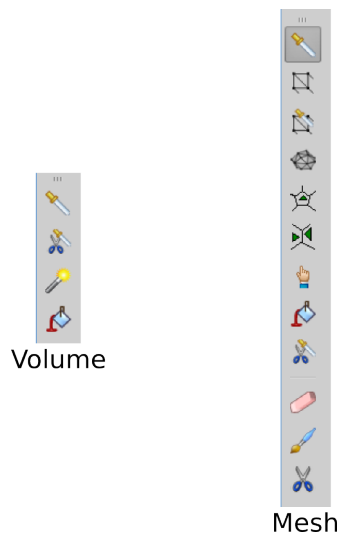
Fig. 2.4: Interaction toolbars

To use the tools of the intefaction toolbar (Figure 2.4), you need to press the Alt key while clicking (see *Configuring the Alt key under Linux*).

The volume tool bar

Pick volume label Set the current label to the one the user click on.

Delete volume label Pick a label on a 3D stack and erase it (e.g. replace it with zeros). It is done using the *Stack.Fill Label* process, setting the new label to *0*.

Pixel edit This tool allows you to change pixels values on the work store of a stack. What the pixels are changed to is decided on the *Stack Editing*.

Fill volume label Pick a label and replace it with the current label. This is using the process *Stack.Fill Label*, setting the new label to the active one.

The mesh tool bar

Pick label Set the current label to

Mesh select Select vertexes on the active mesh or the cutting surface. The selection is one by drawing a rectangle: every vertex within the rectangle will be selected. To use it, the points of the mesh or cutting surface must be visible. By default, at every click, the selection is replaced. To avoid this, press *Shift* while clicking.

Mesh label select Pick a label and select all the vertexes having this label.

Select connected area Extend the current selection to all vertexes connected to it.

Add new seed At each click on the mesh, this tool selects a new seed and label the triangle (e.g. all three vertexes) with it. The seed will be used while the mouse move until the button is released.

Add current seed Set the label of the triangles you click on to the current label.

Grab seed When co-segmenting surfaces, this grab the seed from the other surface and apply it to the triangles being clicked on.

Fill label Replace the label clicked on by the current one.

Delete picked label Delete all vertexes having the label clicked. The deletion only happen when the mouse button is released, giving the opportunity to select more than one label by moving the mouse, keeping the left button pressed.

Erase selection Erase the label of all the selected vertexes.

![brush icon] **Fill selection** Set the label of all selected vertexes to the current one.

![scissors icon] **Delete selection** Delete selected vertexes

2.2.5 Tab Area

Main tab
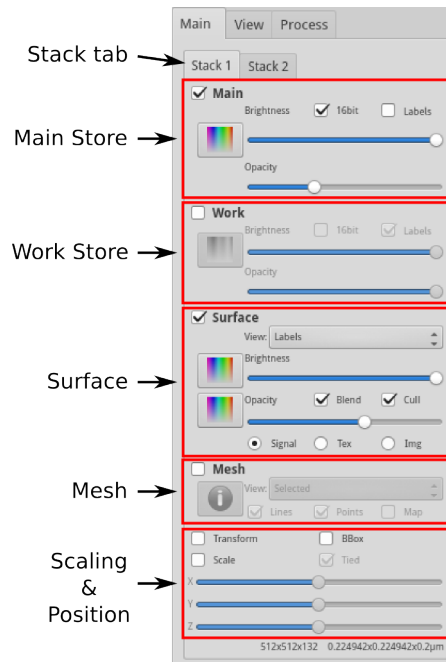
Fig. 2.5: Main tab

The main tab (see Figure 2.5) controls what is displayed and how.

The **stack tab** select the current stack: you can check this looking at the bottom right corner of the window.

The **main and work store** areas are identical:

- the main check box controls their visibility

- the ![transfer function button] button is used to control the transfer function

- the 16 bits check box controls whether the image is displayed using the full 16 bits or only 8 bits. If rendering is too slow, you can try to un-check it.

- the labels check box indicate whether the image is interpreter as signal intensity or labels.

- the opacity and brightness sliders are rather self-explanatory

The display of the mesh is part in two: the surface and the mesh itself.

The **surface** area controls the surface representation of the mesh:

- the top ![transfer function button] button controls the transfer function for the mesh's signal

- the bottom button controls the transfer function for the heat

- the view combo box controls what is displayed on the surface: signal, label, heat or parent label.

- the bottom three radio buttons control whether the surface displays its own information (Surface), the texture attached to it (tex) or the part of the 3D image intersecting the surface (Img).

The **mesh** area control the display of the wireframe representation of the mesh:

- The view combo box allows to select what part of the mesh is visible. Options are: selected, all, cells and border. Cells show the border of cells and border the border of the mesh (in case it is an open surface).

- It is possible to display the lines or the points of the mesh. Note that to select the mesh, it is necessary to display its points.

- The **Map** check box show the labels of each cells as text above the image itself.

- At last the button gives information about the whole stack and mesh: size, resolution, number of vertices, range of the signal, what has been computed, ...

The **scaling and position** area controls global properties of the stack:

- The stack has two position matrices: normal and transformed. The **Transform** check box control which one is displayed / modified. Processes should always change the transformed position.

- The **BBox** check box control whether the bounding box of the image is displayed.

- The **Scale** check box control whether the display size of the stack and mesh is the original one, or the one scaled by the sliders.

- The three sliders control the scaling of the whole stack (including the mesh) along the local X, Y and Z axis. If the **tied** check box is ticked, the three sliders are forced to the same value.

View tab
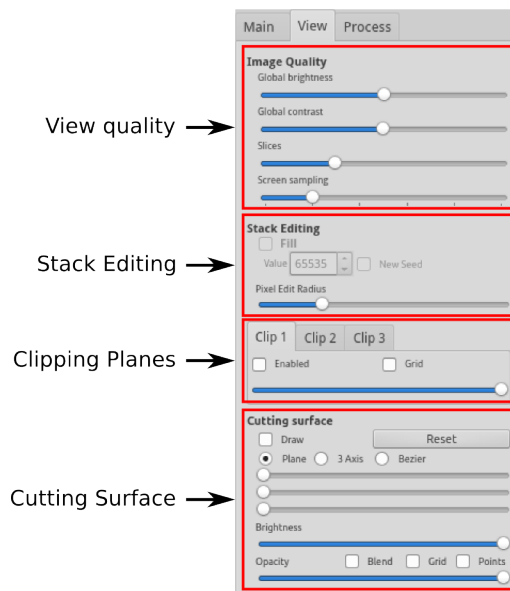
Fig. 2.6: View tab

The view tab (see Figure 2.6) is split in 3 sections: image quality, stack editing and clipping planes.

Image Quality

The view tab is split into 4 boxes. The **image quality** box controls global rendering properties. Beside the common brightness and contrast, you can set the number of slices used to render 3D images: the more slices, the better the rendering, but also the slower. To get a feel for what it does, do not hesitate to descrease it to the minimum (on the left). The last parameter is the **sampling**. When the view moves, instead of rendering the full view, LithoGraphX degrades the image by rendering only a fraction of the pixels. This allows for a precise rendering when the image is still while keeping a fluid movement. If you change this slider, check the result by moving the view.

Note: Most sliders can be reset to their default position by double-clicking on them!

Stack Editing

The second box is the **stack editing** box. First, you will notice it is grayes: it will become available when the current store is the work store of a stack (otherwise, you cannot use the *Pixel Edit* tool anyway). The main control here is the slider, which decides how large the editing pencil is. The size is in screen coordinate, so how much of the object you are editing will also depend on the level of zoom. The default behavior of the pixel edit tool is to erase voxels (e.g. set them to 0). You can choose instead to fill the voxels with other values. Depending on the store (e.g. whether it contains labels or intensity), you will be able to set the value used to fill the stack, or choose to create a new seed at every click.

Clipping Planes

LithoGraphX provides 3 pairs of clipping planes. Each pair are maintained parallel to allow for easy slicing of the objects. To use the clipping planes, first show the grid, checking the **grid** check box. Select the clipping plane in the *Interaction box* and move it wherever you need before checking the **Enabled** box. It is usually easier to move it in place before enabling it.

Process tab

LithoGraphX has three kinds of *Processes*: Stack, Mesh and Global. The three first tabs each list one of the categories.

Within these tabs (see Figure 2.7), processes are organised in folders, which can be open and closed by clicking on the triangle on their left. When a process is selected, its parameters appear in the bottom part of the tab. Be aware that some processes will also provide a dialog box to help you with the parameters. Although the parameters written there will always be taken into account.

The last tab contains tasks. Tasks are ordered collections of processes, usually organized to make protocols easier to follow. Tasks can group processes of all types. They can be saved and loaded from files.
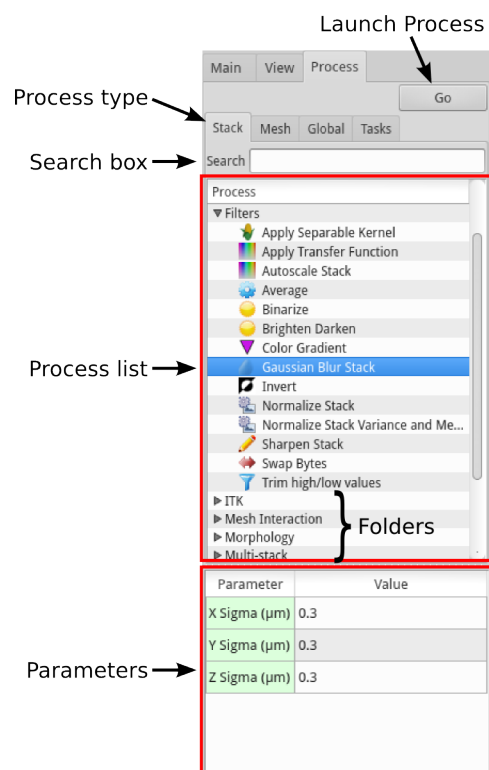
Fig. 2.7: Process tab

Tips for Data Collection

Data collection for use in 3D image processing softwares can be a little different than for other uses. In most cases, people optimize data collection so that individual slices look good to the human eye. This is not always the best for 3D data analysis and visualization. The following are some tips to help get the most out of the images:

- **16 bits images**. If possible collect 16 bits per channel, instead of 8 bits. Although the pictures may look no different, the 16 bits images will have higher dynamic range, and it will be easier to extract features in darker areas of the image. If the microscope only supports 12 bits, that will still be better than 8 bits.

- **More slices, less averaging**. A common technique used to improve the image quality on confocal microscope is frame or line averaging. In this mode, the microscope takes each line/frame many times (typicalle 2 or 4 times), and creates an image by averaging each pass. Although this improves the image quality, it also increase the acquisition time, leading to problems such as: bleaching, tissue damage, movement during acquisition, ... and to avoid these, a common technique is to reduce the number of slices taken. When acquiring data for 3D image processing software such as LithoGraphX, it is recommended to favour increasing the number of slices rather than averaging. The noise will be reduced later using neighboring slices.

- **Dynamic range**. When acquiring images, it is best to maximize their dynamic range: that is the difference between the brightness and the darkest pixel. To help you optimize this, most microscope's software have a mode in which the saturated and black pixels are marked. When in that mode, first set the offset down until you see a lot of black pixels, and then increase it slowly to reduce their number until only a few are left. Then, increase the gain until many pixels are saturated, and decrease it slowly until only a few pixes are saturated. If you are interested in the geometry of cells, it is a good idea to saturate the cell walls slightly more: that will avoid issues with drops in signal and the exact shape will still be found at the middle of the saturated areas.

- **Start and end slices shouldn't contain any sign of your sample**. It is tempting to start the acquisition when the sample is just visible, and stop just before it disappear. This is a bad idea: the reconstruction algorithms need the extra information that "there is no more tissue here" (or at least, not of interest) to reconstruct their geometry properly. So make sure you start a few micron before your sample, and stop a few micros after.

- **Use Fiji to get TIFF files**. Most microscope formats are proprietary. The Loci Bioformat group has reverse engineered the formats of the most popular microscopes and made plugins for ImageJ. We recommend Fiji, which is a distribution of ImageJ that contains these, along with many other useful plugins. Even if your microscope generates TIFF images, you might want to open them with the Bioformat plugins to extract the thickness of the slices. LithoGraphX only recognise ImageJ generated TIFF and OME TIFF for this.

- **If voxel sizes are wrong**. The TIFF format has no standard way to specify the voxel size in Z. ImageJ defines its own attributes for this, and the OME tif format another one. LithoGraphX recognises both formats. But some manufacturers provide their own TIFF file, with a different specification of the slice thickness. In this case, LithoGraphX will be unable to read it. You will need to use the Stack process `Change Voxel Size` in the `Canvas` folder and specify the `z` resolution.

Tutorials

You will find here a series of tutorial to accomplish various tasks. Each tutorial is given with a sample dataset and a LithoGraphX task to get you started.

The datasets used in these tutorials can be found there: http://www.lithographx.org/data-gallery

4.1 Loading and visualizing data

For this tutorial, download this dataset, which we used while working on *[Yoshida2014]*.

4.1.1 Loading the data

There are four ways to load data in LithoGraphX.

1. The simplest is to open the data directly with LithoGraphX. If you installed it correctly, data files native to LithoGraphX are associated with it. If you right-click on such a file you should have en entry to open it with LithoGraphX. Once launched, the data is simply loaded in the Stack 1, on the Main store for stacks.

2. Once LithoGraphX is open, to load more data or replace existing data, the next simplest way is to use drag&drop. If you drop the file on the *interaction area*, it will be loaded in the currently active stack. For 3D images, you can also drop them on the main and store boxes in the *main tab* to load it at a particular location.

3. You can use the menu to load either a LithoGraphX project, a stack or a mesh file. There are different menus for the various targets.

4. You can directly use the loading processes: they are in the `System` folders of the Stack, Mesh and Global process tabs.

Try the various means to load the files `160112_4.tif`, `160112_4.mgxm` and at last the project file `160112_4.lgxp`.

4.1.2 Tuning the visualization

For this part, load only the TIFF file `160112_4.tif`.

Opacity and basic transfer function

When loading this dataset, you will see a very opaque object filling in almost the whole image. This is an embryo of *Arabidopsis thaliana* within its ovule, and we need to change the visual properties to start to see it.
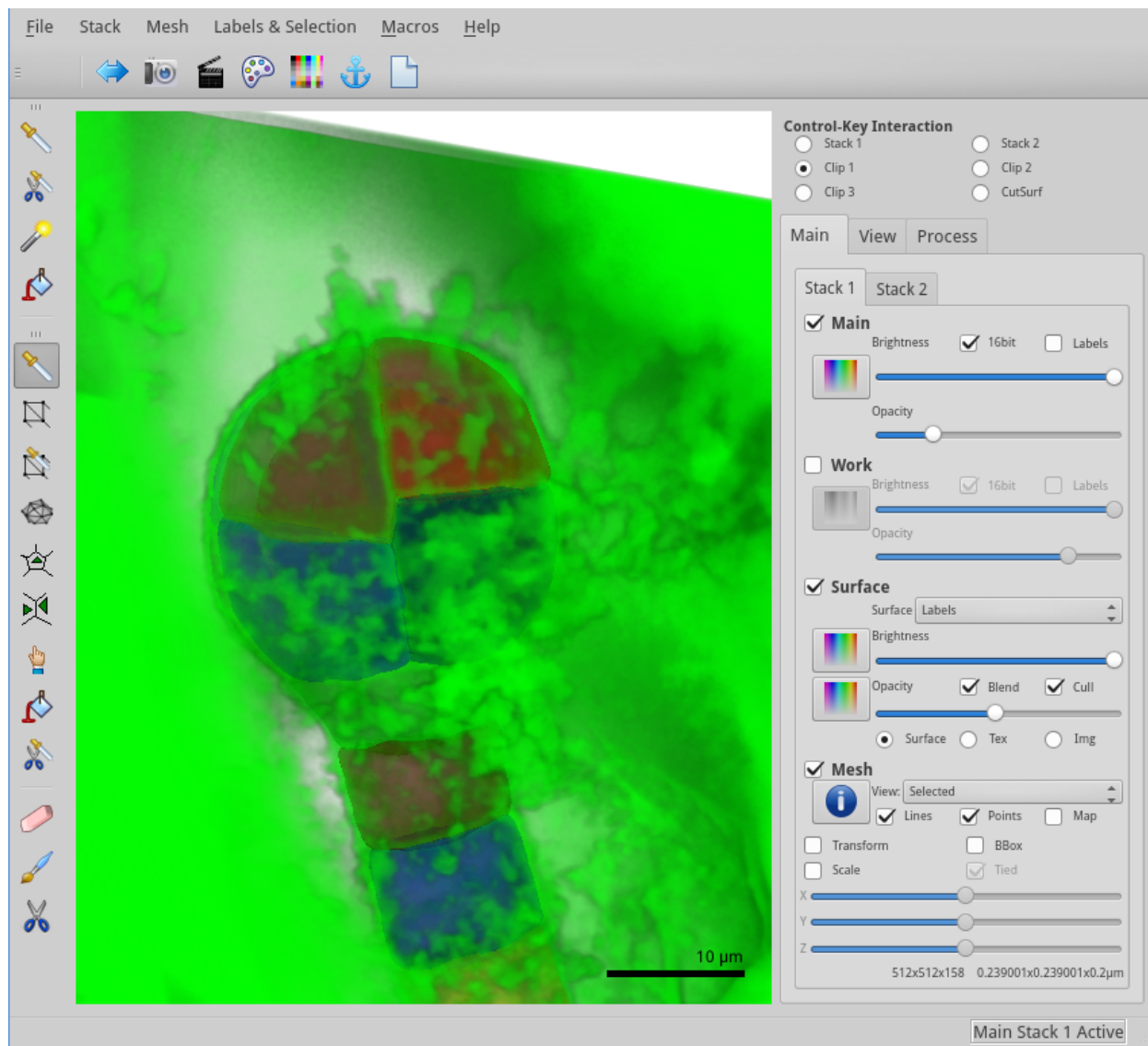
Fig. 4.1: 16-cells embryo of *Arabidopsis thaliana*. Image: Saiko Yoshida, Weijer's lab.
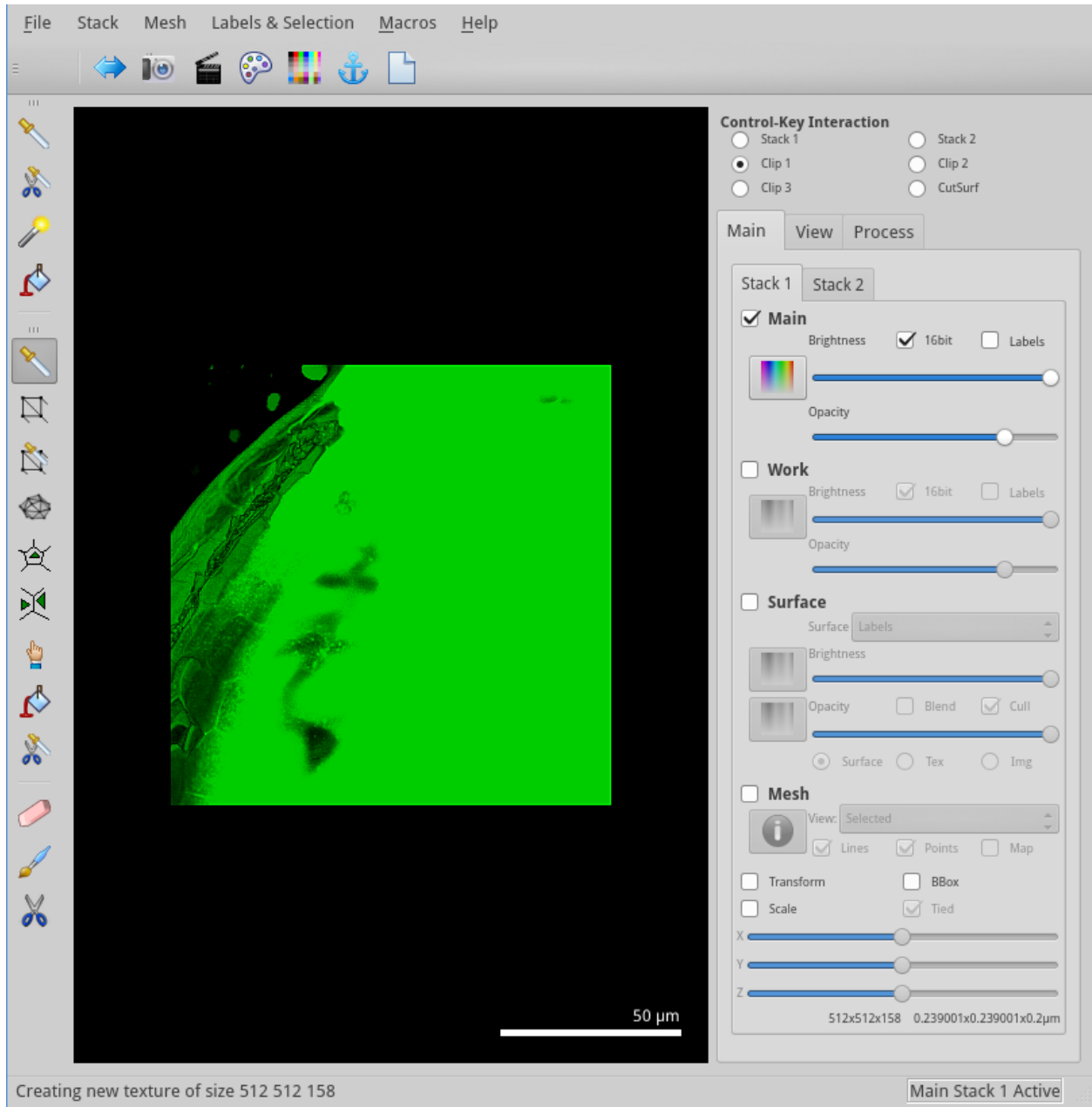
Fig. 4.2: Initial view of the dataset.

But first, click the [icon] button to see the transfer function (see Figure 4.3).
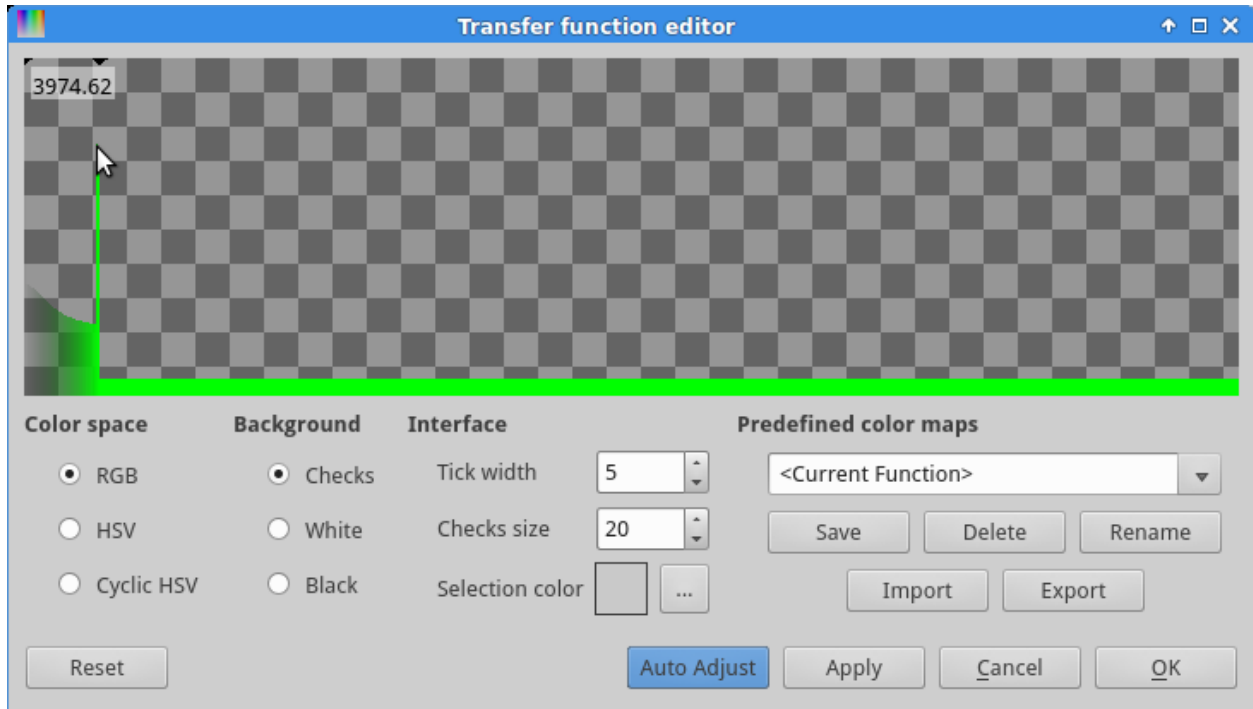
Fig. 4.3: Transfer function of the object: note how only the low values are used.

We can see that the signal is entirely on the left of the spectrum! This is because the image is a 12 bits image. The number on the top-left indicate the value under the mouse cursor: a 12 bits image has a maximum possible value of 4095, while a 16 bits image can have values up to 65535. For further processing, we will want to expand the range of values. This will allow us to work with the representation with more ease, and will ensure fewer variations in the parameters of the data processing. **Process** `[Stack]Filters/Autoscale Stack` The process `Stack.Autoscale Stack` in the *Filters* folder, will do just that: rescale the stack to use the whole range of possible values. This process has no parameter and will simply modify the current store. You will notice a change in colour from green to cyan: this is because the result has been saved in the work store of the stack 1, and this store is by default cyan. We can look now at the transfer function of the work store (see Figure 4.4).

Now, we can see the stack uses the whole spectrum of values, which will be important later.

But to see the embryo, you need to decrease the opacity of the store (see Figure 4.5).

Surface and Mesh

Now, load the mesh, check the **Blend** check box and adjust the opacity.

To look at the signal, we need first to create some. For example, compute the Gaussian curvature on each point:

Process	[Stack]Signal/Project Mesh Curvature
Parameter	**Value**
Type	Gaussian
Neighborhood (m)	2
Autoscale	Yes

Then, click on the upper [icon] button to change the transfer function of the signal (this is the second such button in the surface box). In the dialogue box, select the "French flag" transfer function (see Figure 4.7).
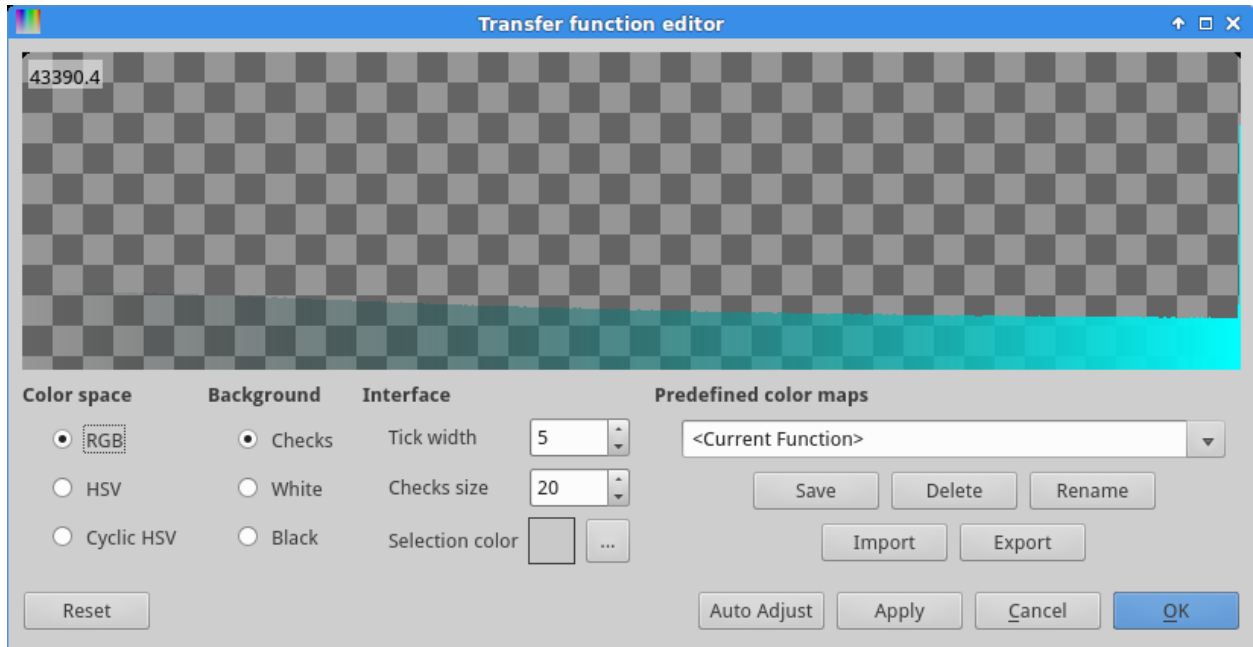
Fig. 4.4: Scaled transfer function.

Another property of the mesh is the **heat**. While the signal is per point, the heat is per label. THe main way to create a heat map is to use the process: **Process** `[Mesh]Heat Map/Heat Map` When you launch the process, a fairly large dialogue box appear. Other tutorials will explain in more details what it is for. For now, simply change the heat map type fo "Volume" and make sure the visualization is "Geometry" (see Figure 4.8).

After validation, the surface visualization automatically changed to "Heat" and displays the result (see Figure 4.9).

To change the transfer function, you can use the lower button.

Changing the colours

The main colours can be changed using the button (see Figure 4.10).

For example, to re-create the image at the top of this page, change the background to white, and the legend and scale bar to black by double-clicking on the colour and use the dialogue box to choose the colour. To use if fully, you might also want to change the editing pixels and clipping planes colour to something darker.

Rendering speed

If rendering is slow, there are a number of settings in the *View tab* that can be helpful in the view quality box:

- **Slices**: changes the number of slices used to render the volume. The more the slower but also the better the quality. You can try to push it fully on the left to see how artefacts look like.

- **Screen sampling**: to speed up rendering while keeping a good image quality, LithoGraphX can reduce the image quality only when moving. This parameters sets the sub-sampling done while moving.
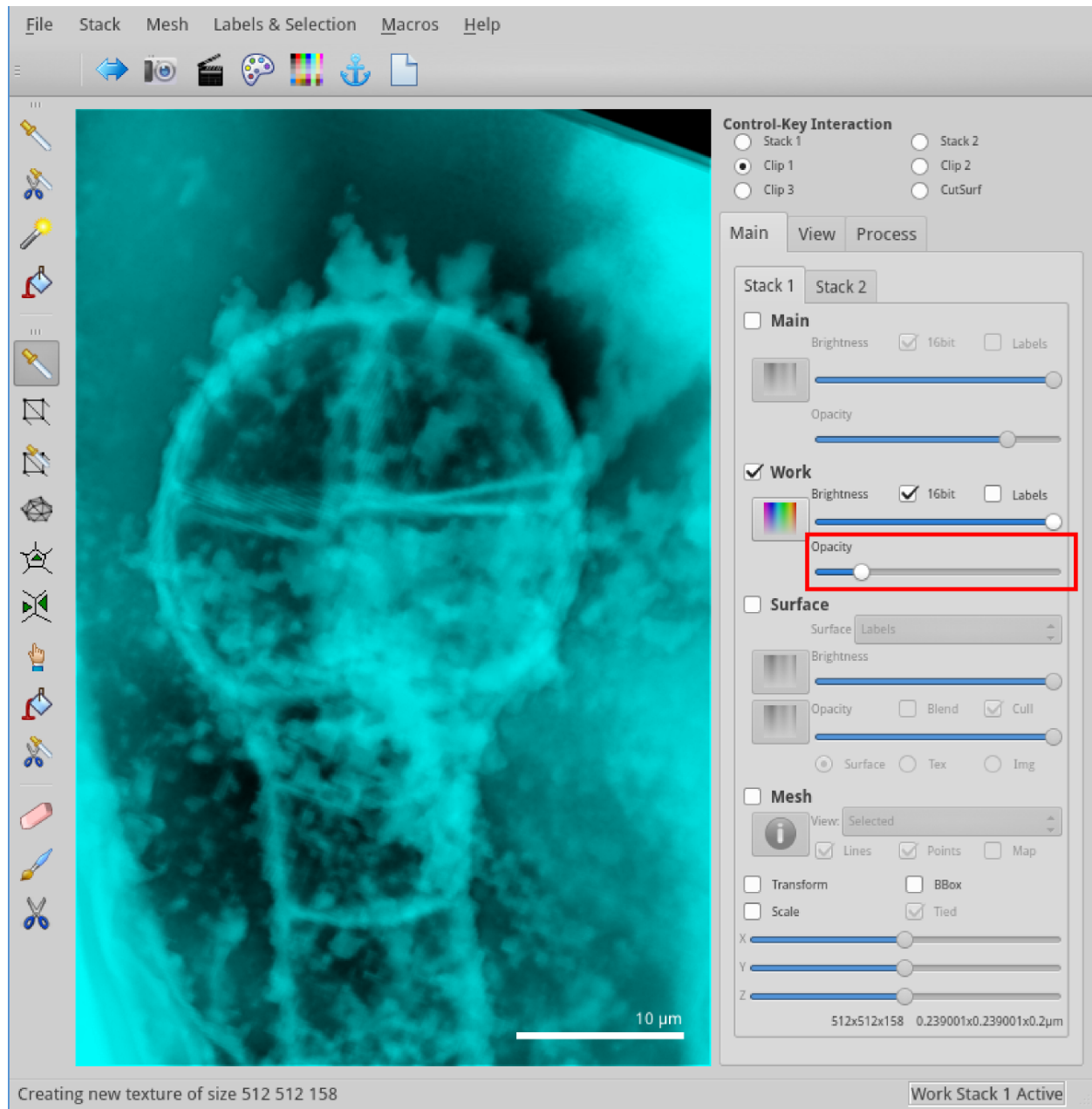
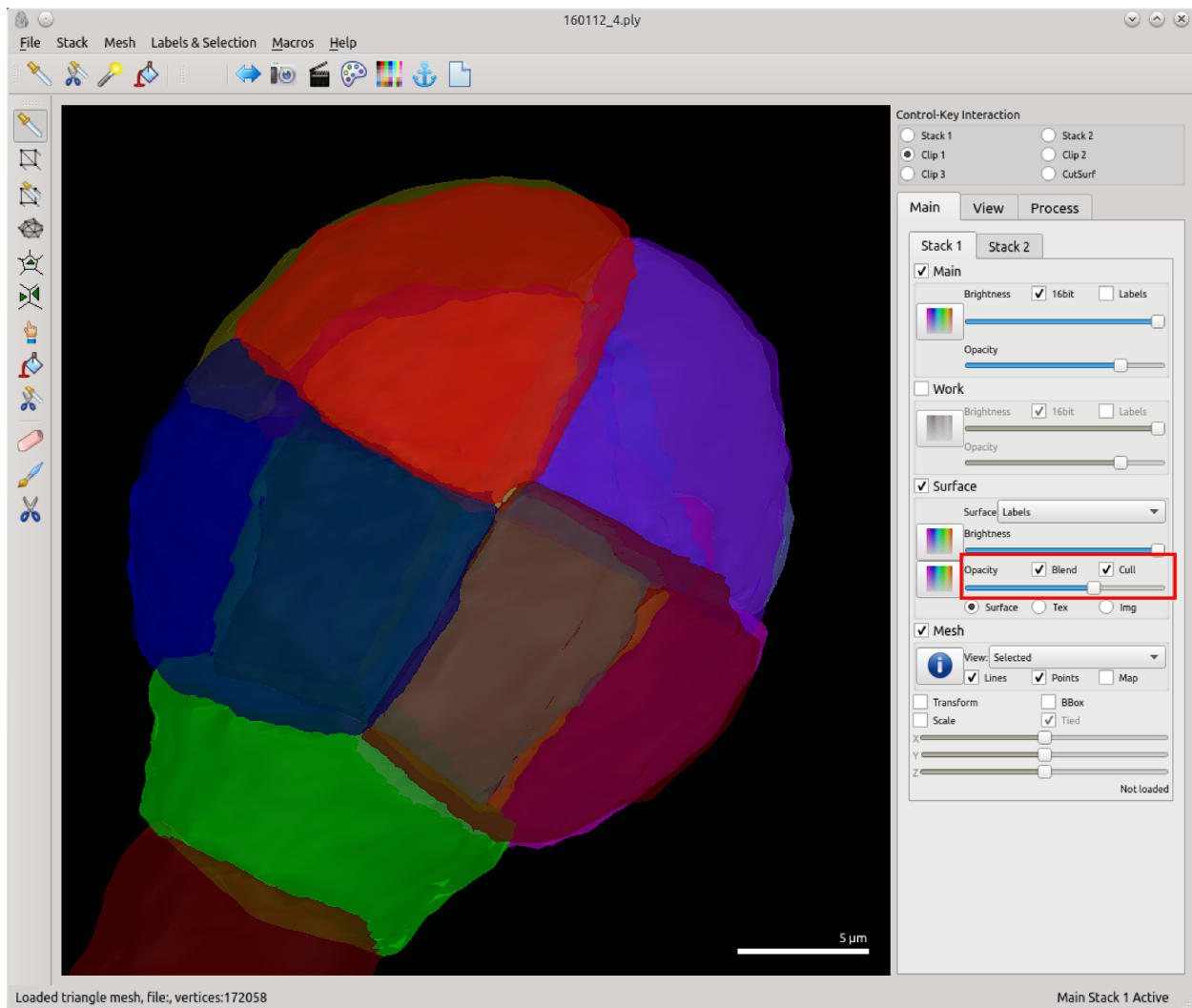Fig. 4.5: We need to adjust the opacity to reveal the embryo.
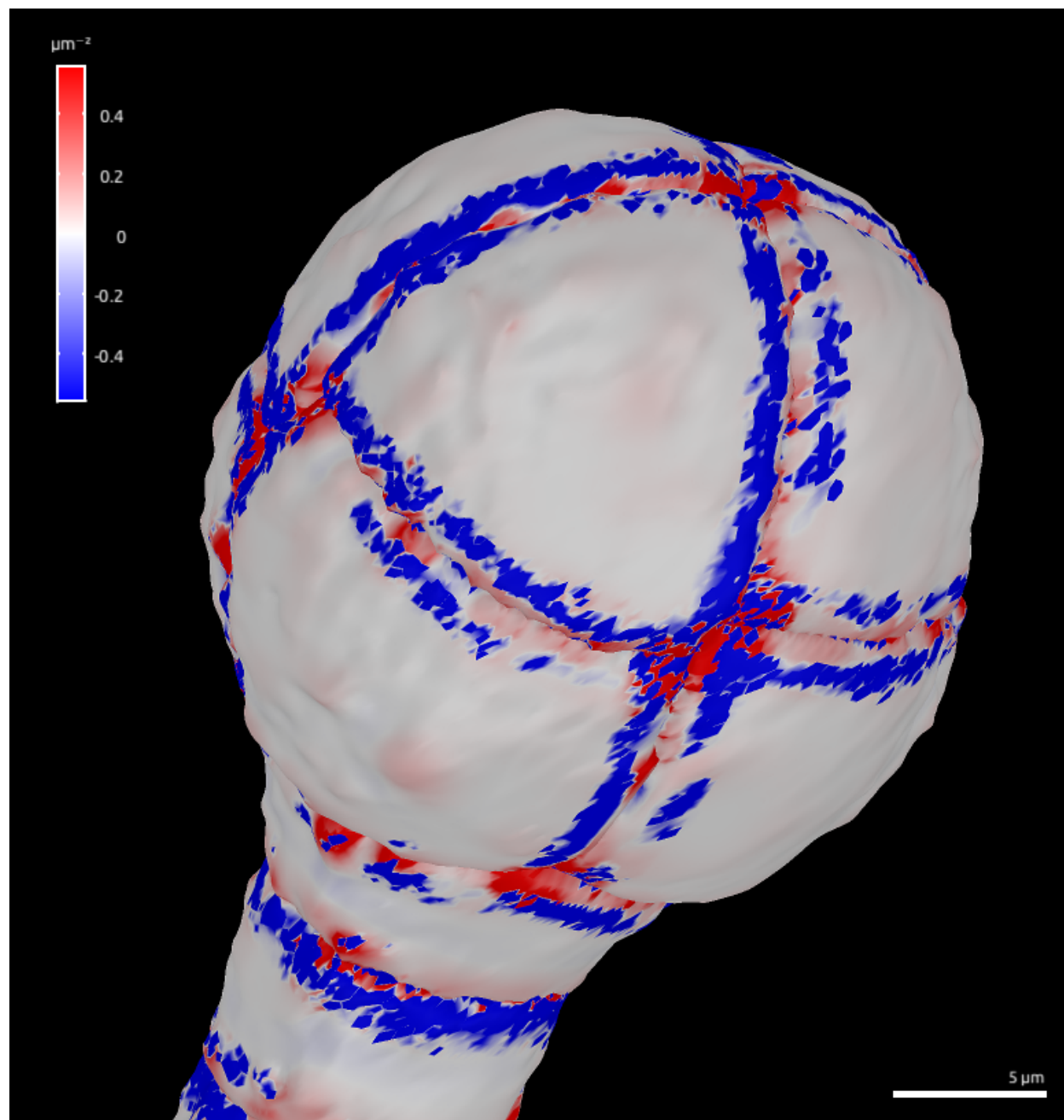
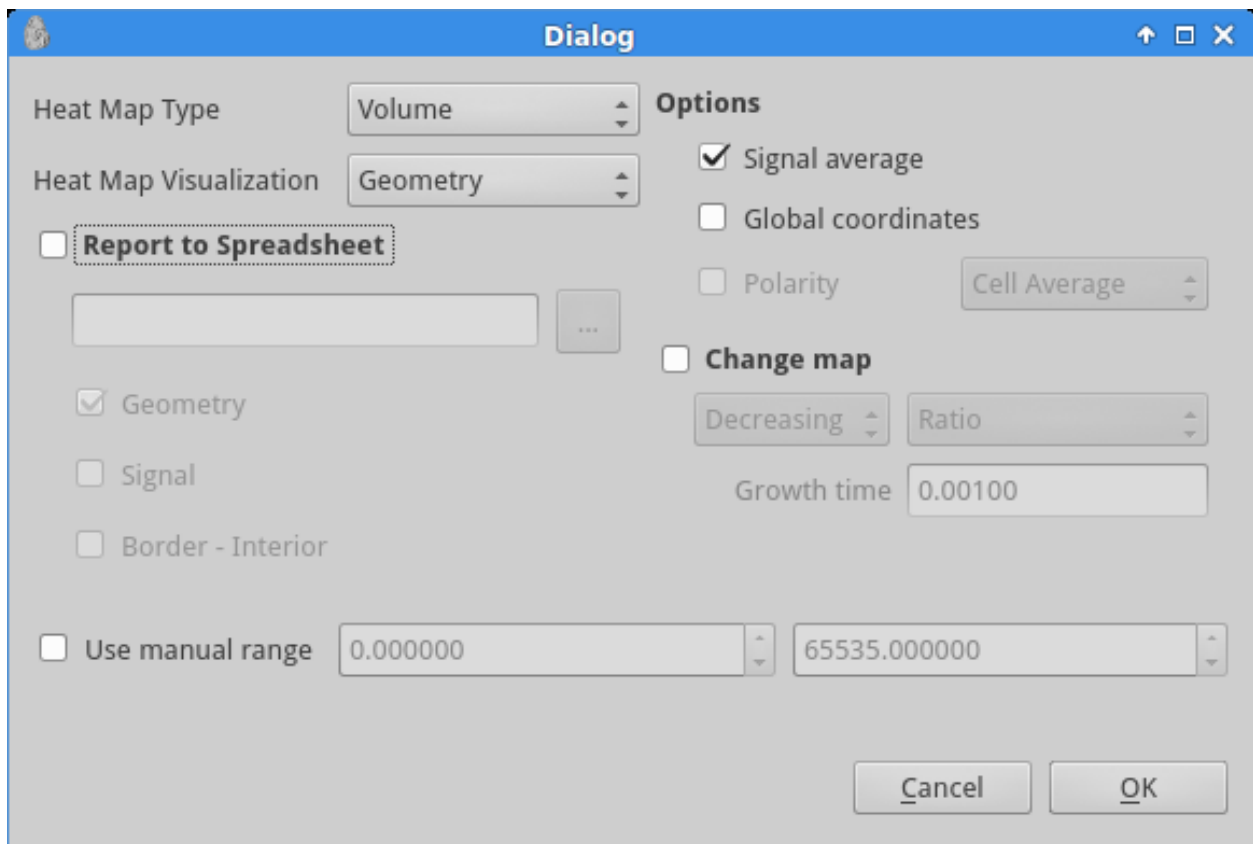Fig. 4.6: Mesh covering the cells in the Embryo.

Fig. 4.7: Curvature of the mesh.

Fig. 4.8: Dialog box of the Heatmap process.

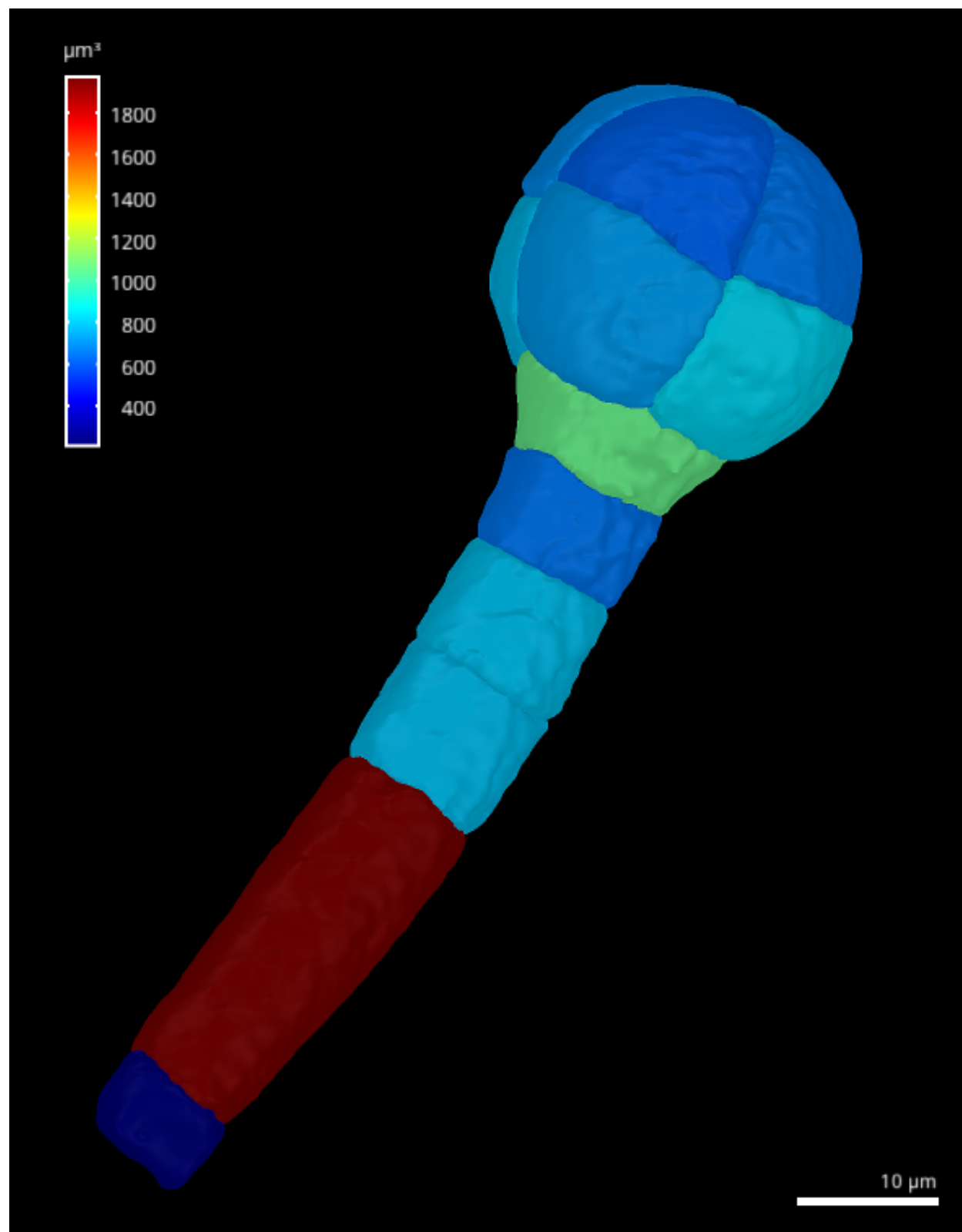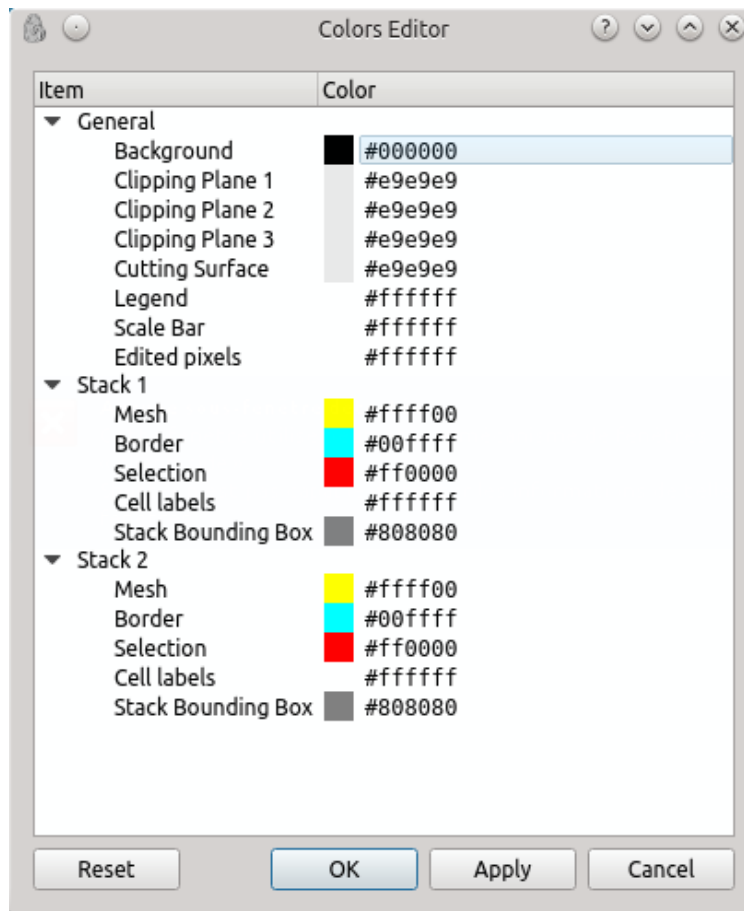Fig. 4.9: Head map of the cell volumes.

Fig. 4.10: Colors Editor

Transfer function

TODO

Inside the object

Clipping Planes

Cutting Surface

Processes

4.2 Data Visualization (2)
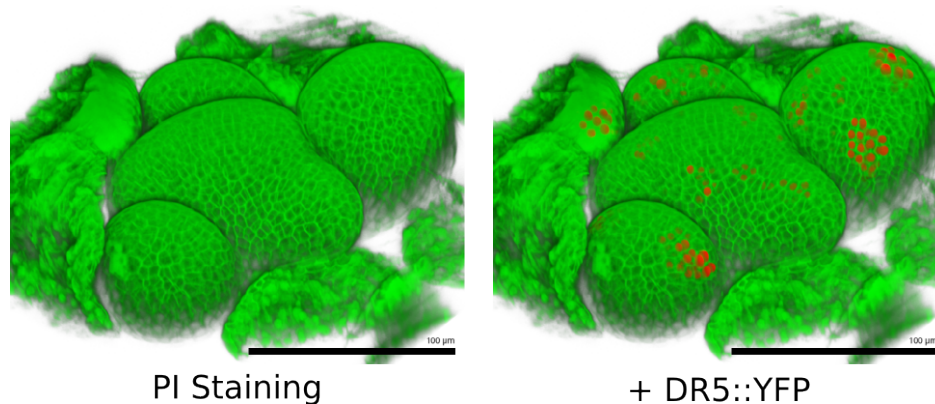
Fig. 4.11: Shoot apical meristem of *A. thaliana*, cell walls stained with PI and with DR5::YFP fluorescent marker. Image: Agata Burian

4.2.1 Introduction

The purpose of this exercise is to introduce the interface of LithoGraphX, learn to segment cells in 3D and work with the heat map.

Note that you will find a lot of useful information in the `Help` menu, including a user manual, a documentation of all the existing processes, and the mouse and keyboard interactions.

LithoGraphX has a fixed number of objects that exist at any given time. They are organized in two `Stacks`, each stack containing two images and a mesh. The stack defines the size, resolution and position of the images it contains. The two images are called the `Main` and `Work` store: the main store is where the data is loaded by default, while the work store is where the result of any processing will be put. Only one store can be **active** at any given time. Which is active depends on the visibility of the stores and which stack tab is currently active. It is however, indicated on the right of the status bar, at the bottom of the main window.

Hint: In the beginning, always check which store is active before launching processes.

Each stack also contains a triangular mesh. To each vertex of the mesh is attached a signal and a label. The label however, is interpreter by triangle: if at least two vertices of a triangle have the same label, the whole triangle is given

this label. In case of tie, the triangle is considered un-labelled. At last, to each label a heat may be associated. A mesh can be seen either as a continuous surface, displaying the signal, label or label's heat, or as a wireframe mesh, or both at the same time.

In LithoGraphX, loading, saving, processing, analysing is done through **Processes** (in fact, most menu items are simply shortcuts to processes). There are three kind of processes: Stack, Mesh and Global. Stack processes can only modify the stacks and stores, the Mesh processes can only modify the meshes and Global processes can modify anything.

4.2.2 Loading a stack

Once LithoGraphX is started, the simplest way to open a new image stack is to drag&drop the file (tif) onto the drawing area. You can also use the `Stack` menu to open a file to a particular store.

4.2.3 Loading a second channel

To load a second channel, you can either use the menus, or again drag&drop a file. However, instead of dropping onto the drawing area, you can drop the file onto the `Work` store area (see *Main tab*).

4.2.4 Clipping planes
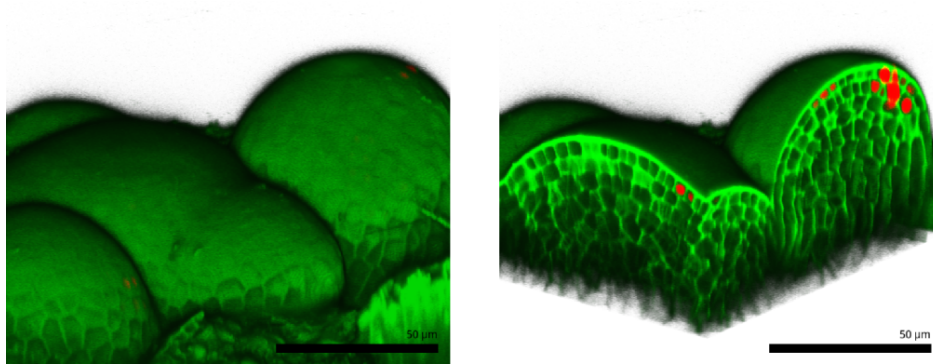

Without clipping   With clipping

You should now have a nice image the SAM, But you cannot easily see inside! To help with this, LithoGraphX has few tools. The first one is three pairs of clipping planes (see Figure~ref{fig:clipping}). To work with the clipping planes, they must be enabled in the `View` tab. Each pair of clipping plane has its own tab, from which the user can enable it, show a grid to see its position when not enabled, and set the distance between the two planes. To move the planes, it must be selected in the main tab in the `Control-Key Interaction` box. As the name suggest, the clipping planes are then moved by pressing the `Control` key and use the mouse in the same way as for moving the camera.

Try using the clipping planes to see where the DR5::YFP signal is with respect to the cell layers.

4.2.5 Cutting Surfaces

Another way to see the insides of a volume is to use cutting surfaces. Like the clipping planes, the cutting surface need to be enabled in the `View` tab, at the bottom. The cutting surface can take one of three forms: a single plane, three orthogonal planes, or a Bézier surface. When enabled, the part of the visible volumes that intersect the chosen surface will be drawn, ignoring their brightness and opacity settings. Instead, the brightness and opacity of the surface is usedfootnote{To see a volume only on the cutting surface, push its opacity slider all the way to the left, making it completely transparent.}.

Cutting plane   3 cutting planes   Bézier surface
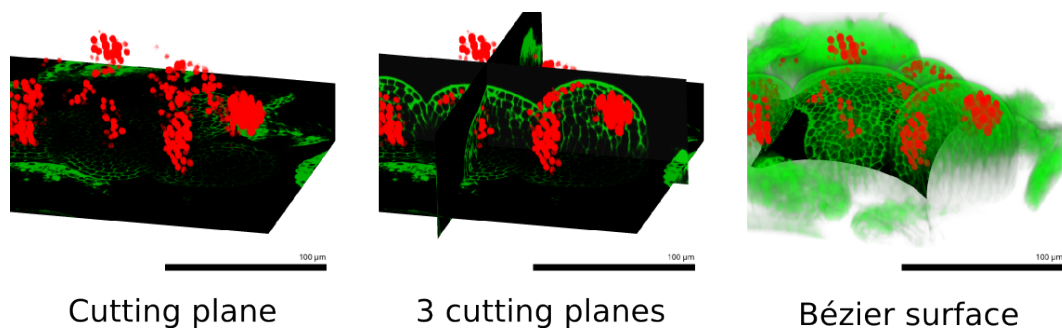
As for the clipping plane, you can move the cutting surface is it is selected in the `Control-key interaction` box.

At last, for the Bézier surface, if you show the grid and make sure points are shown, you can select and move them using the mesh selection tool.
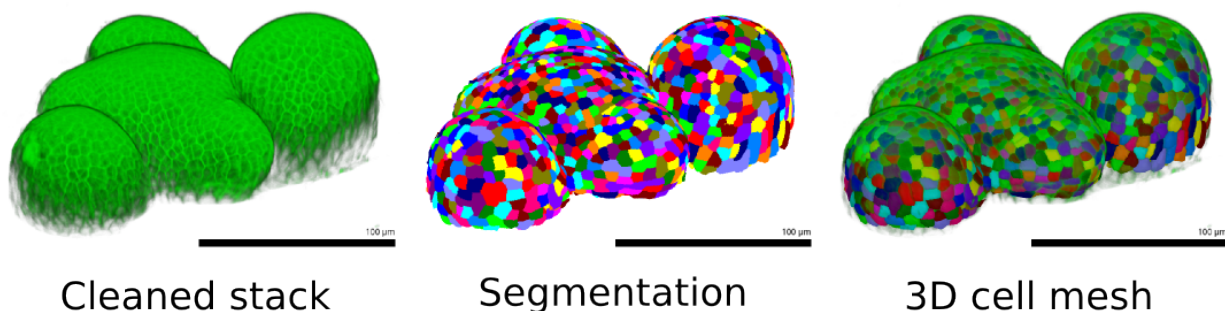
4.3 Segmenting Cells in 3D

Cleaned stack   Segmentation   3D cell mesh

Fig. 4.12: Shoot apical meristem of *A. thaliana*, segmented in 3D

In this section, we are going to see how we can extract the 3D shape of cells in images of the shoot apical meristem of *Arabidopsis thaliana*, and see how we can use an extra channel with the segmentation. To start, download this dataset, provided by Dr. Agata Burian.

4.3.1 Loading the data

For this tutorial, you can start either by loading the file `Arabido_DR5-YFP_pi.tif` or `Arabido_DR5-YFP_pi_cleaned.tif`.

4.3.2 Cleaning the data

If you have loaded `Arabido_DR5-YFP_pi.tif`, you will see in addition to the shoot apical meristem a few lateral organs, including those partially removed to acquire the image. To extract only what we want to see, we need to clean the image.

1. Copy the stack to the work store. At the same time, we will autoscale the stack's intensity. This is an important step as it makes to rest of the process much easier.

> **Process** `[Stack]Filters/Autoscale Stack`

2. Select the *pixel edit tool*. In the *View tab* make sure `Fill` checkbox in the *Stack Editing* area is not checked. Then, pressing the Alt key, you should see a circle above the drawing area. This circle marks the cylinder that will get erased. Click to erase.

3. When you are happy with your cleaning, save the stack!

Note: If the area to erase is too large, nothing will change until you release the mouse button.

Hint: You can also do the cleaning on the segmented stack. In this case, another useful tool is the *fill volume tool*. Ensure no label is currently selected by clicking on the *current label* button. Then click on the labels you want to erase.

4.3.3 3D Cells Segmentation

1. Make sure the stack is in the Main store.

2. Blur the stack

Process	`[Stack]Filters/Gaussian Blur Stack`
Parameter	**Value**
X Sigma (m)	3
Y Sigma (m)	3
Z Sigma (m)	3

3. Segment the stack:

Process	`[Stack]ITK/Segmentation/ITK Watershed Auto Seeded`
Parameter	**Value**
Level	1500

4. Remove external cells:

Process	`[Stack]Segmentation/Erase at Border`
Parameter	**Value**
Distance (m)	0
X	Yes
Y	Yes
Z	Yes

5. Extract the cell shapes:

Process	`[Mesh]Creation/Marching Cubes 3D`
Parameter	**Value**
Cube Size (m)	1
Smooth Passes	3

Hint: Use the clipping planes to scan through the volume and make sure the segmentation worked correctly in the depth of the tissue.
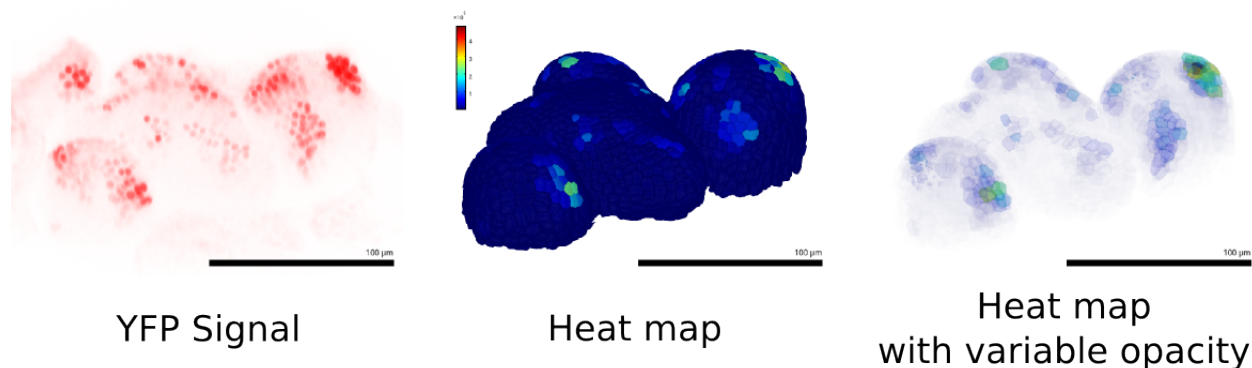

Fig. 4.13: Signal quantification on 3D mesh

4.3.4 3D Signal Quantification

1. Load the YFP channel in the work store of the stack 1. To do that, the simplest is to drag&drop the file onto the *work store area*.

2. Launch the `Heat map` mesh process:

   > **Process**   `[Mesh]Heat Map/Heat Map`

3. In the dialog box, select the `Volume` heat map type and the `Total signal` visualization and make sure `Signal average` is not ticked.

The result is, per cell, the total amount of signal present in the YFP channel.

Note: This is **not** sufficient for a proper signal quantification. This is only qualitative. For a proper quantification of the signal you need a reference channel.

4.4 Segmenting Cells in 3D (advanced)

For this first tutorial, we are going to look step by step into the 3D segmentation of the cells of the embryo of Arabidopsis.

First, download the dataset from BitBucket here: dataset.

4.4.1 Loading the dataset

1. Launch LithoGraphX

2. Drag & drop the file `160112_4.tif` into the interaction area of LithoGraphX

3. Adjust the opacity as we have seen in the *previous tutorial*.

You should see the embryo like in Figure 4.14.
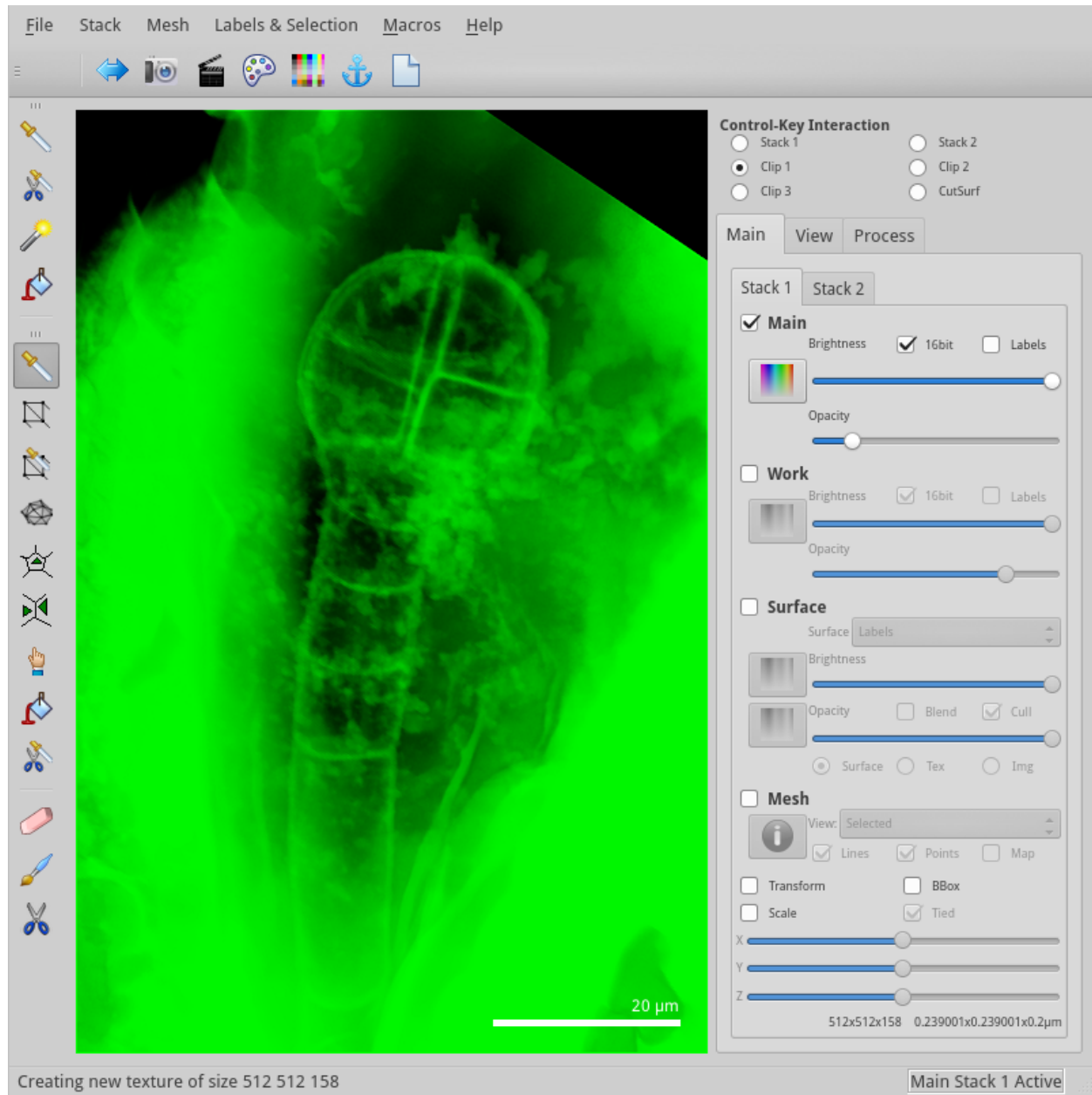
Fig. 4.14: Loaded stack with the embryo

4.4.2 Extracting the embryo

As you can see, in addition to the embryo, we have a lot of extra tissues. There are a number of tools we can use to remove most of it. We will first use the clipping planes and then the pixel editing tool.

Trimming with clipping planes

1. Go to the *View tab*

2. Show the grid of the first clipping plane.

3. Make sure `Clip 1` is selected in the *Interaction box*.

4. Ajusting the thickness of the slice, and moving the clipping plane with the control-key pressed, place the clipping planes to include the embryo (see Figure 4.15)

5. Enable the clipping planes

6. Clip the image itself:

   Process   `[Stack]Mesh Interaction/Clip Stack`

7. Adjust the transfer function of the work stack: click on the ▦ button of the work store and click on the auto-adjust button. At last, adjust the opacity.

8. Disable the `Clip 1` from the *View tab*.

Using the Pixel Editing tool

1. Select the *pixel editing tool*

2. Ensure the active stack is the `Work Stack 1`

3. In the *View tab*, make sure the `Fill` check-box in the *Stack Editing* box is not checked.

4. Remove pixels by pressing the Alt key and clicking. All visible pixels within the drawn circle will be erased! After a few minutes of erasing, you should get something like Figure 4.16.

5. When you are done, copy the result to the main store.

   Process   `[Stack]Multi-stack/Copy Main to Work Stack`

Troubleshooting

- When using the pixel edit, depending on the view angle (e.g. the number of voxels that would need editing), the edition may be done only when you release the mouse button.

- If you make a mistake, there is no undo! Unless you copy the stack back to the main store, where you will be able to access it again.

   Process   `[Stack]Multi-stack/Copy Main to Work Stack`

- Don't panick if you find it hard to move around. This is normal at the beginning, but you will get used to it. A commonly forgotten useful shortcut allows to turn the object in the plane of view with Left+Middle mouse click (look at the Mouse and Keyboard help).
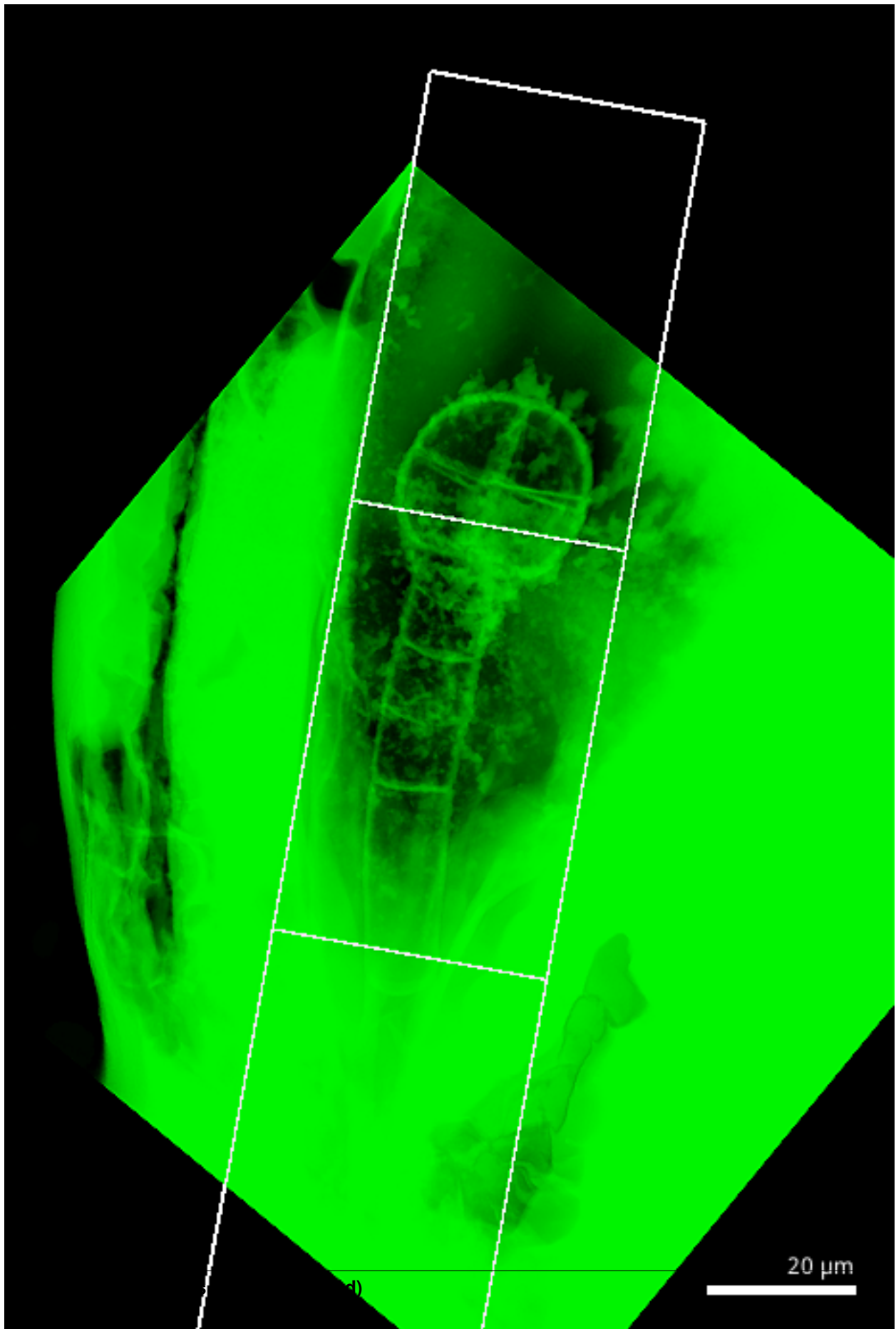
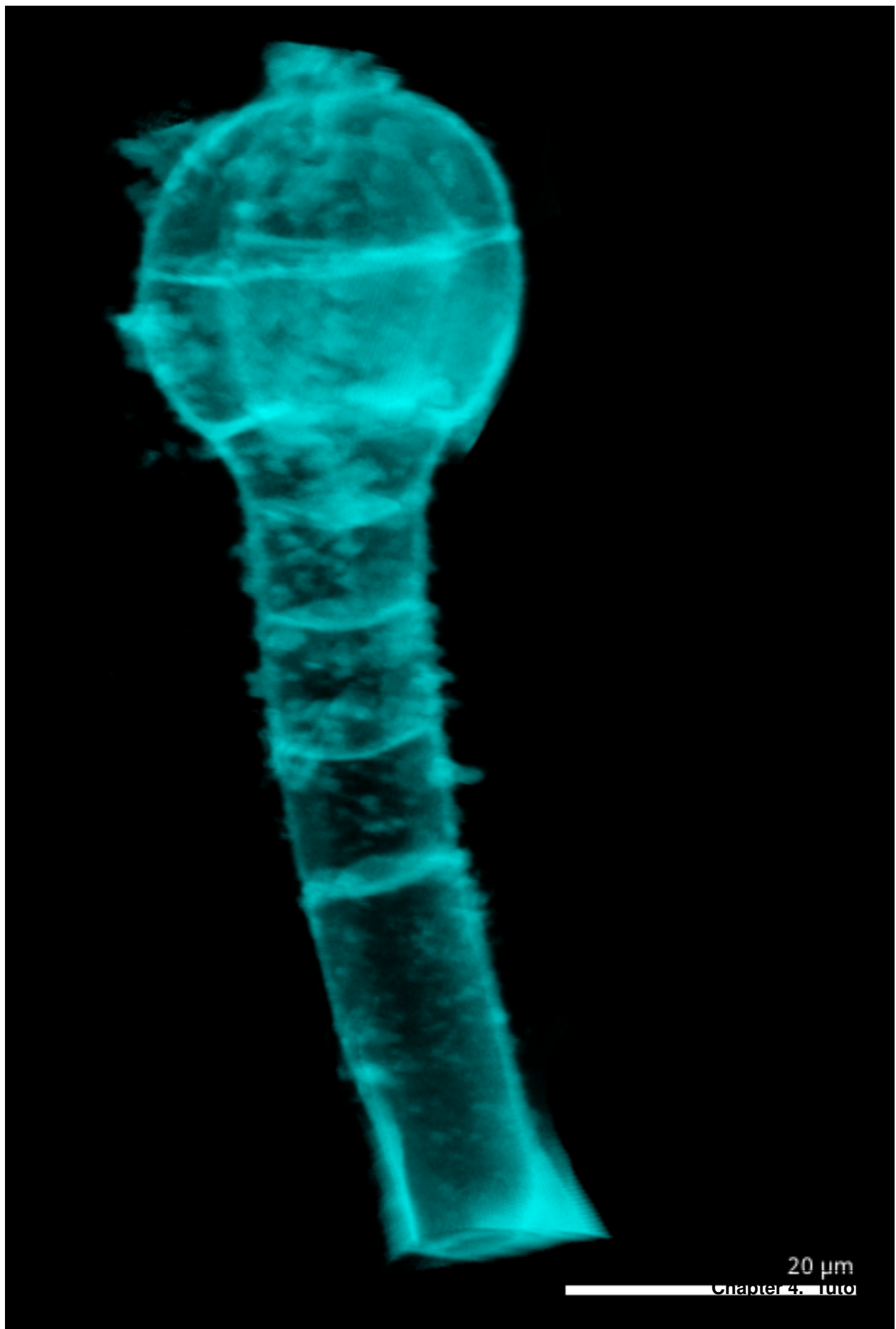Fig. 4.15: Place the clipping planes to include just the embryo itself.

Fig. 4.16: Cleaned stack

4.4.3 Segmentation

Now we have a nice, isolated image. We are ready for the segmentation part.

1. To make it easier and more reproducable, we need to scale the values to use the whole range. Run the process `Stack.Autoscale Stack` from the `Filters` folder. If you changed the transfer function, you might need to re-adjust it.

2. The image is too noisy to be segmented as is. You need to blur the image:

Process	`[Stack]Filters/Gaussian Blur Stack`
Parameter	**Value**
X Sigma (m)	0.3
Y Sigma (m)	0.3
Z Sigma (m)	0.3

3. We will now segment the stack. The `Level` parameter is the most important one: it tells the algorithm the level of background noise to consider. Remember the values range from 0 to 65535, so 1500 is quite small. But if you didn't autoscale the stack before, it may become very large compare to the actual maximum intensity!

Process	`[Stack]ITK/Segmentation/ITK Watershed Auto Seeded`
Parameter	**Value**
Level	1500

4. Use the `Fill picked label` tool to erase the outside cell: make sure no label is selected, and Alt-click on the outside cell to delete it. There might be some other small cells to erase.

At that stage, the segmentation is mostly good, but some cells are too large. We could of course edit by hand the initial stack to improve, but we are going to use another means, more in line with what LithoGraphX is good with: interaction between mesh and stack.
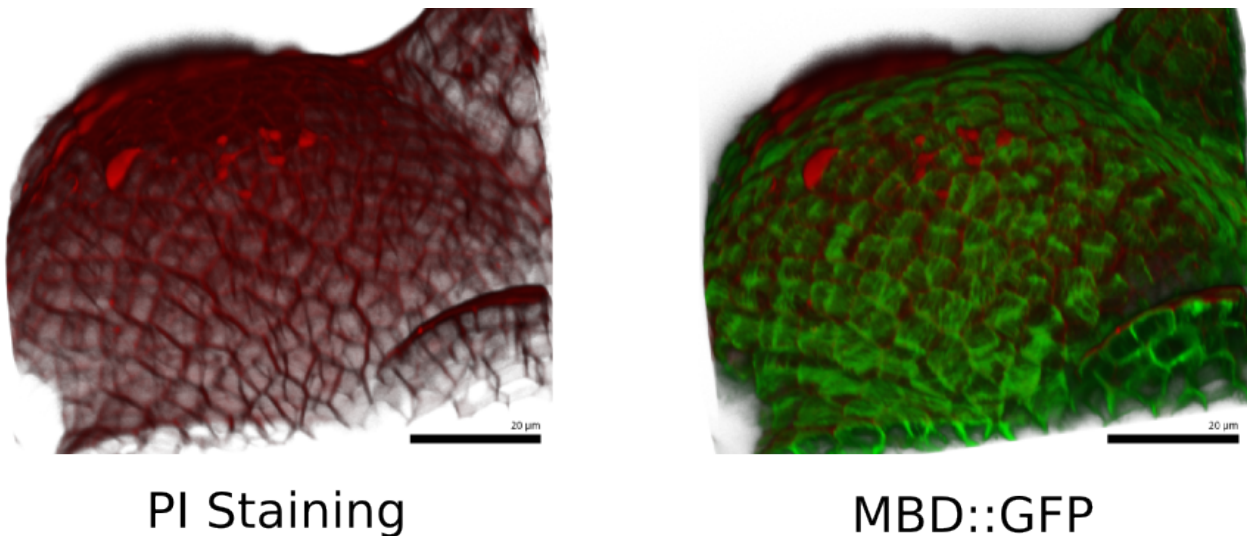
4.5 Segmenting cells in 2D

Fig. 4.17: Shoot apical meristem of A. thaliana, with microtubules

In this section, we are going to see how we can extract the 2D shape of cells in images of the shoot apical meristem of *Arabidopsis thaliana*, and see how we can use a marker of cortical microtubule to compute their orientation based on the method published in *[Boudaoud2014]*. To start, download this dataset, provided by Dr. Agata Burian.

To segment the cells in 2D, we need to proceed in two phases:

1. Extract the surface of the meristem

2. Segment the cells on that surface

4.5.1 Extracting the surface of the meristem

See figure 4.18 A-D.

1. Blur the stack slightly:

Process	`[Stack]Filters/Gaussian Blur Stack`
Parameter	**Value**
X Sigma (m)	0.5
Y Sigma (m)	0.5
Z Sigma (m)	0.5

2. Extract the surface, using all the default arguments:

Process	`[Stack]Morphology/Edge Detect`

3. Erase the bumps due to dead cells by hand, using the `Pixel Edit` tool

4. Extract a coarse surface:

Process	`[Mesh]Creation/Marching Cubes Surface`
Parameter	**Value**
Cube size (m)	3

5. Select the bottom and sides with the selection tool: make sure the `Mesh` check-box is ticked and `View` is on `Selected`. Then, orienting properly the meristem, press the `Alt` key and click and drag a rectangle to select the vertices to remove. You can extend the selection by pressing the `Shift` key while clicking. For a demonstration of what needs to be done, look at this video.

6. We now need to refine the surface. For this, we will alternate smoothing and refining the mesh. Check the terminal for the number of vertices after subdivision.

 | Process | `[Mesh]Structure/Smooth Mesh` | | Process | `[Mesh]Structure/Subdivide` |
   | --- | --- | --- | --- | --- |
   | **Parameter** | **Value** | | | |
   | Passes | 3 | | | |

Youtube video: `http://www.youtube.com/watch?v=ZbTaYITkObg`

4.5.2 Segment the cells

See figure 4.18 E-F.

1. First, make sure the PI channel, which should be the Main Stack 1 is the active stack. In the Main tab, the `Stack 1` tab should be the selected tab, the `Main` store should be selected and the `Work` store not.

2. Then, project the signal onto the surface. You can play with the mim and max distances to see what happens when they change.
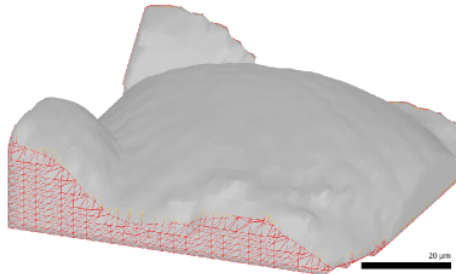
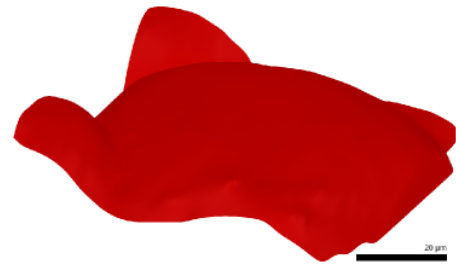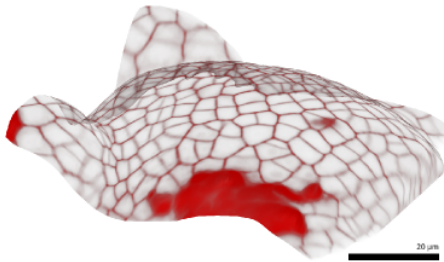 | Process | `[Mesh]Signal/Project Signal` |
   | --- | --- |
   | **Parameter** | **Value** |
   | Min Dist (m) | 1 |
   | Max Dist (m) | 1 |
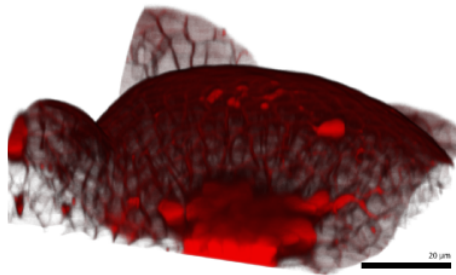
(A) Staining
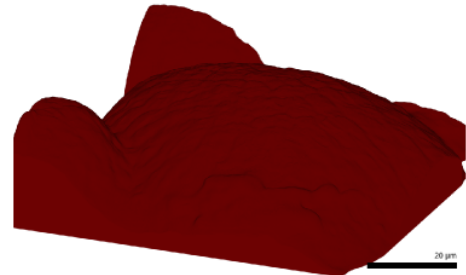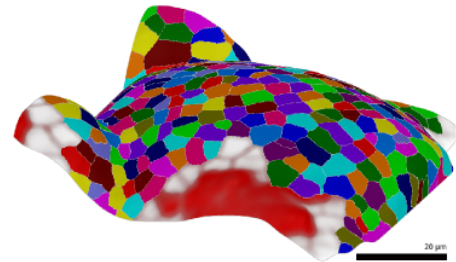

(B) Volume of the meristem

(C) Selection of sides for removal

(D) Smoother surface

(E) Surfaces with signal

(F) Segmented cells

Fig. 4.18: 2D Segmentation of the meristem

3. Now, we are going to segment the cells visible in the signal.

Process	[Mesh]Segmentation/Auto-Segmentation
Parameter	**Value**
Update	Yes
Normalize	No
Blur Cell Radius (m)	1
Auto-Seed Radius (m)	2
Border Distance (m)	1
Combine Threshold	1

4. After a while, you will see outline of segmented cells. In the Main tab, select `Labels` as representation for the surface. And using the 2D bucket, erase the cells above the areas where the staining didn't work.
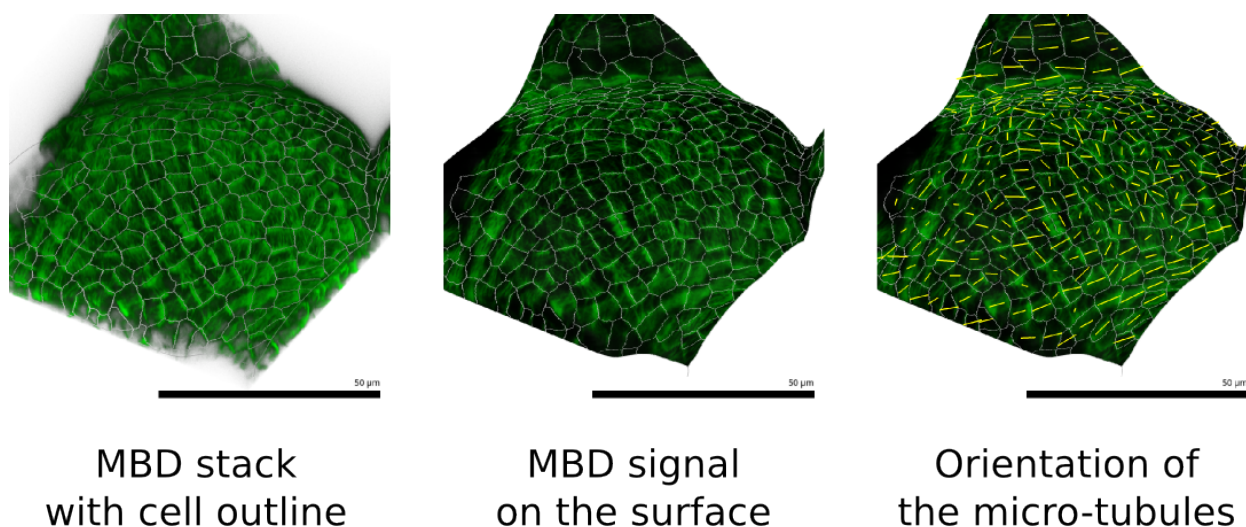
4.5.3 Micro-tubule orientations

MBD stack
with cell outline

MBD signal
on the surface

Orientation of
the micro-tubules

Fig. 4.19: Extraction of the microtubules orientation.

1. Load the MBD stack: just drag and drop the file on the LithoGraphX window.

2. Project the new signal onto the surface. Select a range from *-1 μm* to *1 μm*. You can try different values and see which provide the best contrast.

3. Compute the fibril orientations:

    Process   [Mesh]Cell Axis/Fibril Orientation/Compute Fibril Orientations

4. You can also adjust the way the orientation is displayed:

    Process   [Mesh]Cell Axis/Fibril Orientation/Display Fibril Orientation

4.6 Segmenting cells from a 2D image and automated cell classification

In this section, we are going to see how we can extract the 2D shape of cells from a 2D image of the cross-section of a hypocotyl of *Arabidopsis thaliana*, and see how we can train a cell classifier to automatically label cell types. This
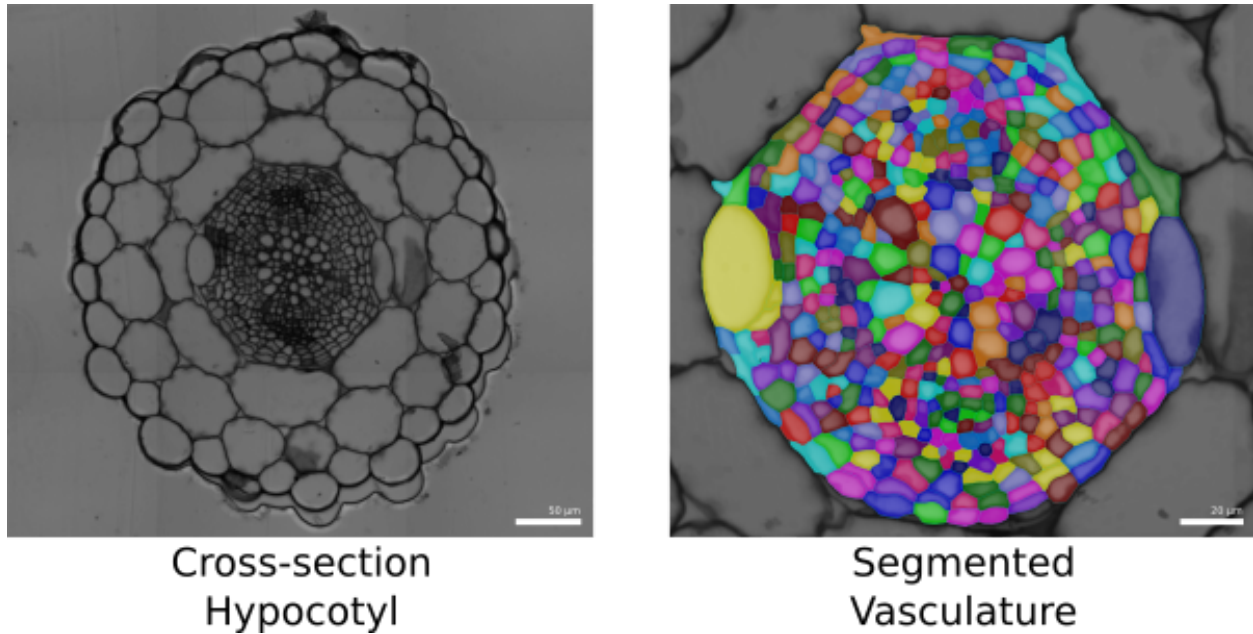
Fig. 4.20: Cross-section of a hypocotyl of A. thaliana at 14 days

tutorial is based on *[BarbierDeReuille.Ragni.InPress]*. To start, download this dataset, provided by Dr. Laura Ragni.

4.6.1 Pre-processing

Although the image is of very good quality, the cells contain a lot of small "objects" that may cause problem when segmenting. In this sense, an object is a contiguous group of pixels whose intensities are all strictly greater and strictly smaller than their neighbors. For the pre-processing, we will remove them:

1. Load the image `Col_14d_63x_bl1_4sti_s1.ome.tiff`

2. Change the transfer function to `Scale Gray` and auto-adjust its range.

3. Re-scale the stack's intensities to maximize its dynamic range:

   > **Process**   [Stack]Filters/Apply Transfer Function

4. Filter the stack using two gaussian blurs with a very small radius (about the half voxel size):

Process	[Stack]Filters/Gaussian blur
Parameter	**Value**
X Sigma (m)	0.1
Y Sigma (m)	0.1
Z Sigma (m)	0

5. Remove all the small objects from the image. This process will remove any object (see above for the definition) whose area (for a 2D image) or volume (for a 3D image) is smaller than some threshold. Note that the shape of the object is irrelevant. As the smaller cells are about *3-4 μm^2*, set the size to *2 μm^2*:

Process	[Stack]Morphology/Sieve Filter
Parameter	**Value**
Size (m²)	4

6. Save this stack as it might be useful later.
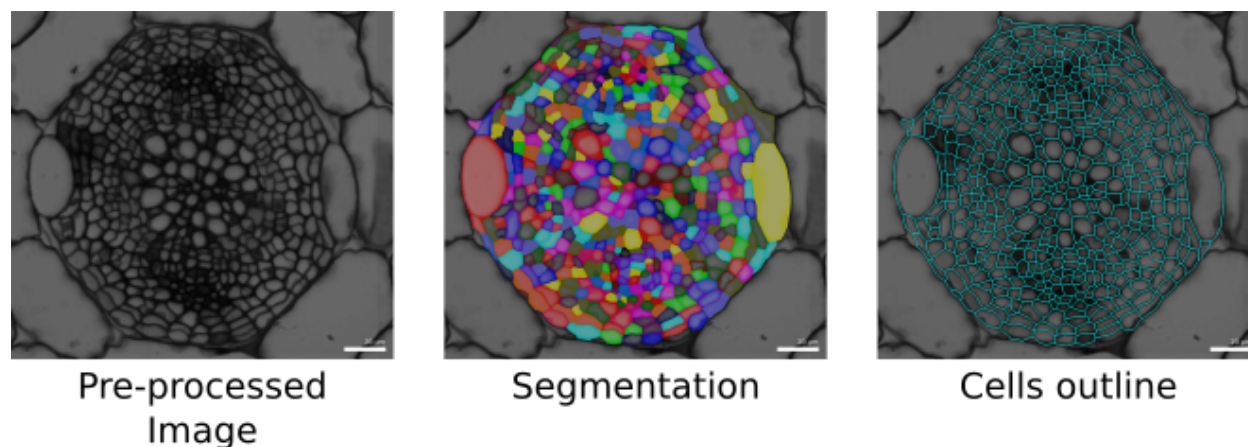
4.6.2 Segmentation of the vascular tissues

Fig. 4.21: Segmentation of the vascular tissues

1. Use the `Segment Section` process in the `Segmentation` folder. This process will blur the cells slightly and use the auto-seeded watershed for the segmentation itself. The auto-seeded watershed requires the cell walls to be bright and the background to be dark, which is the opposite of our images here. The last option of the process is to invert the image before segmentation. Otherwise, use the default options.

Process	[Stack]Segmentation/Segment Section
Parameter	**Value**
BlurX	0.3
BlurY	0.3
Invert	Yes

2. We now need to remove all the segmented parts outside our region of interest. In the image, the vascular tissue is contained in a disc of radius *80 μm*:

Process	[Stack]Segmentation/Remove Labels in Shape
Parameter	**Value**
Shape	Sphere
Center	0 0 0
Size	80

3. Create a cell mesh from the segmentation with a fairly fine description of the cell:

Process	[Mesh]Cell Mesh/Cell Mesh from 2D Image
Parameter	**Value**
Edge Length (m)	1

4. To inspect the segmentation, it is simpler to show the outline of the cells. For this, in the *main tab*, un-check the `Surface` check box and check the `Mesh` check box. For the mesh, select to view the cells (in the `view` combo box) and hide the points.

Correcting the segmentation

With the cell outline visible, you can check the correspondance between the segmentation and the image. Segmentation errors can be broadly classified in three categories:

1. Over-segmentation is when a single biological cell has been identified as many cells by the segmentation..

2. Under-segmentation is when some biological cells have been identified as a single cell by the segmentation.

3. Mis-segmentation is any error that doesn't fall into either previous category.
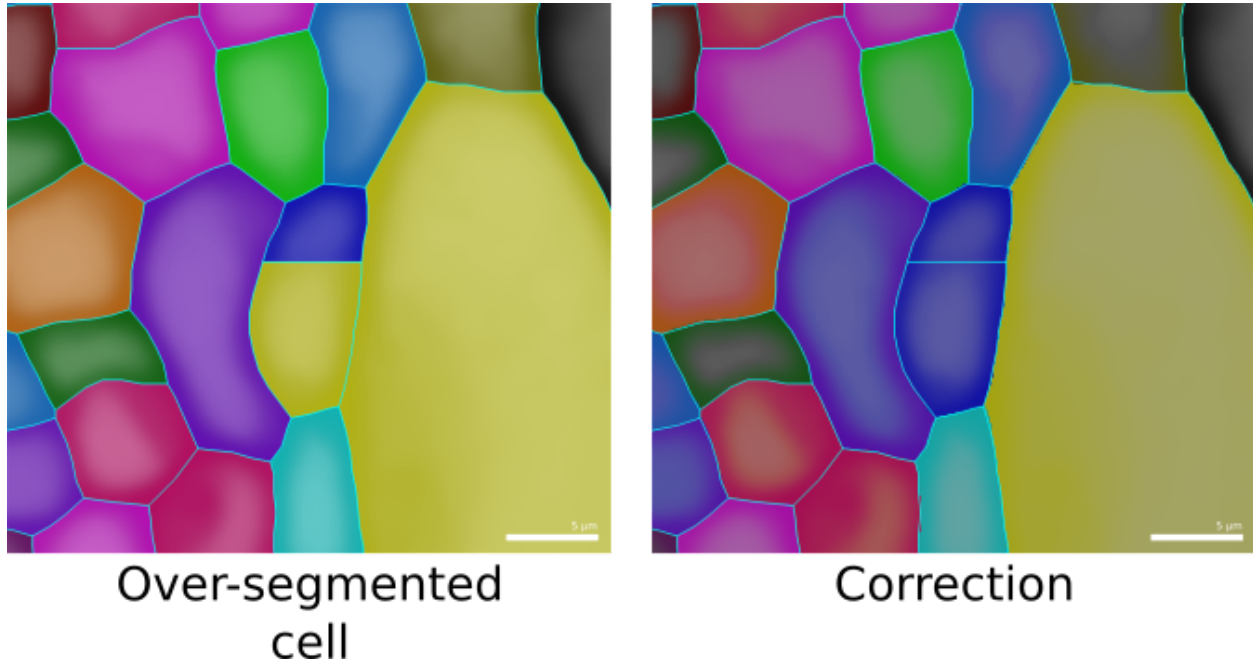
Over-segmentation

Fig. 4.22: Example of over-segmented cell.

Over-segmentations are the easiest error to correct.

1. Show the segmented stack (which should be in the work stack)

2. Using the 3D pipette, select the label of part of the cell

3. Using the 3D bucket, replace all the other labels in the biological cell by the one selected.

4. Re-create the cell mesh

Under-segmentation

Under segmentation is a bit more difficult as it requires local re-segmentation.

1. Erase the under-segmented cell.

2. Select the magic want and, in the *view tab*, in the `Stack Editing` section, select `Fill` and `New Seed`.

3. Place seeds at the center of each biological cells.

4. In the menu `Labels & Selection`, select `New Seed`

5. Using the 3D bucket, flood-fill the outside of the tissue. To flood fill an area, you need to press Shift and Alt while clicking with the left button of the mouse.

6. Swap the main and work stack:

      Process   `[Stack]Multi-stack/Swap Main and Work Stacks`
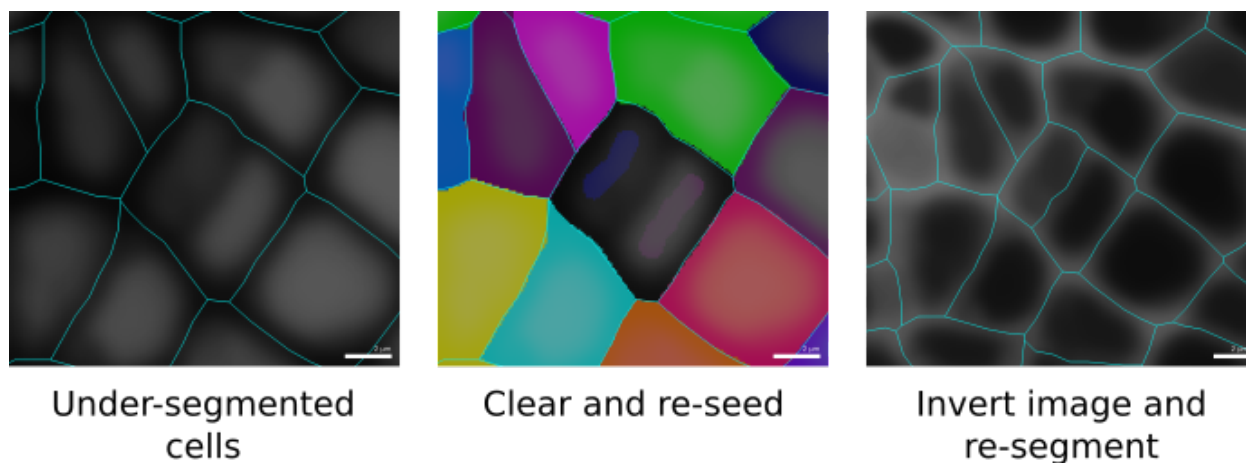
7. Invert the main stack:

4.6. Segmenting cells from a 2D image and automated cell classification   **55**

Fig. 4.23: Example of under-segmented cell.

> **Process** `[Stack]Filters/Invert`

8. Swap the stacks back in place

9. Re-segment the image:

> **Process** `[Stack]ITK/Segmentation/ITK Watershed`

10. Delete the outside cell with the 3D bucket tool.

11. Re-create the cell mesh

12. Optionally you can re-invert the image by repeating steps 6 to 8.

Mis-segmentation

Correcting mis-segmentation is very similar to correcting under-segmentation. The only difference is that all mis-segmented cells should be deleted. Also, mis-segmentation is often due to issues with wall staining. If this is the case, place the seeds close to the cell wall to prevent the re-segmentation from "bleeding"".

4.6.3 Automated cell classification

An automated classifier will try to identify cells from a set of adapted features. LithoGraphX offer a process computing a whole series of features adapted to cells segmented from 2D images. As often, cells will be identified by a numeric identifier, which you need to choose for each cell type. In this protocol, here are the identifiers we chose:

Table 4.1: Cell types identifier

Cell Type	Identifier
Xylem vessel	3
Phloem bundles	4
Cambium	5
Xylem parenchyma	6
Phloem parenchyma	7

Creating the classifier is done in three steps:

1. Label some cells by hand

2. Compute the features

3. Train the classifier

It will be important that all files linked to a given image share a common prefix.
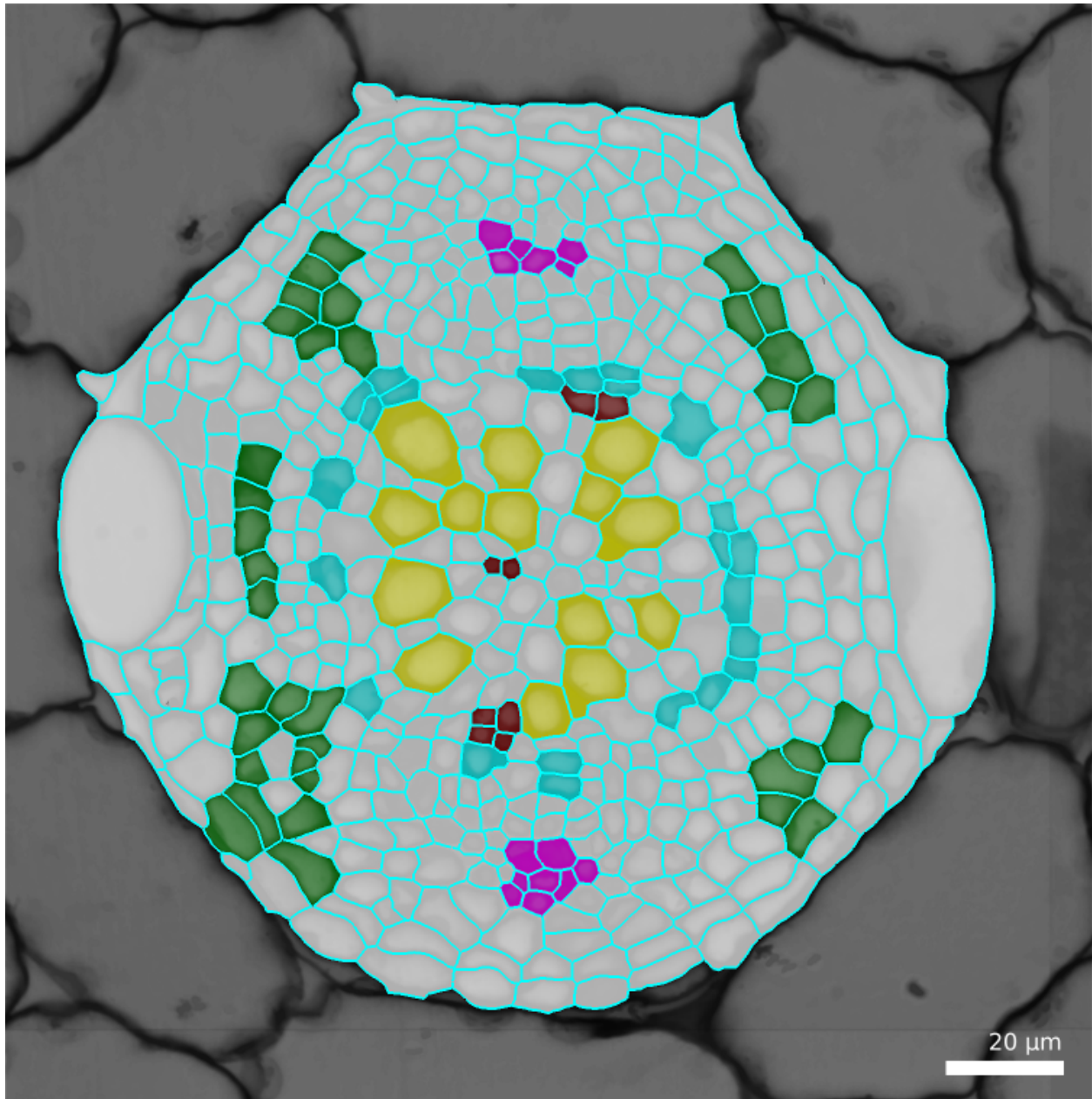
Labelling of the cells

Fig. 4.24: Example labelled cells.

Cells are labelled using the "parents" feature. Except that, instead of giving to a cell the id of its parent, we are going to give it its cell type identifier.

1. Show the surface, selecting the `Parents`

2. To see more easily, check the `Blend` check box and decrease the opacity by moving the slider.

3. In the menu `Labels & Selection` select `Change current label` and enter one cell type identifier.

4. Using the 2D bucket, Alt-Left click on some of the cells of that identify

5. Save the mesh file.

6. Repeats steps 3 and 4 for each cell type, marking each time some of the cells and trying to mark as many of all the cell types.
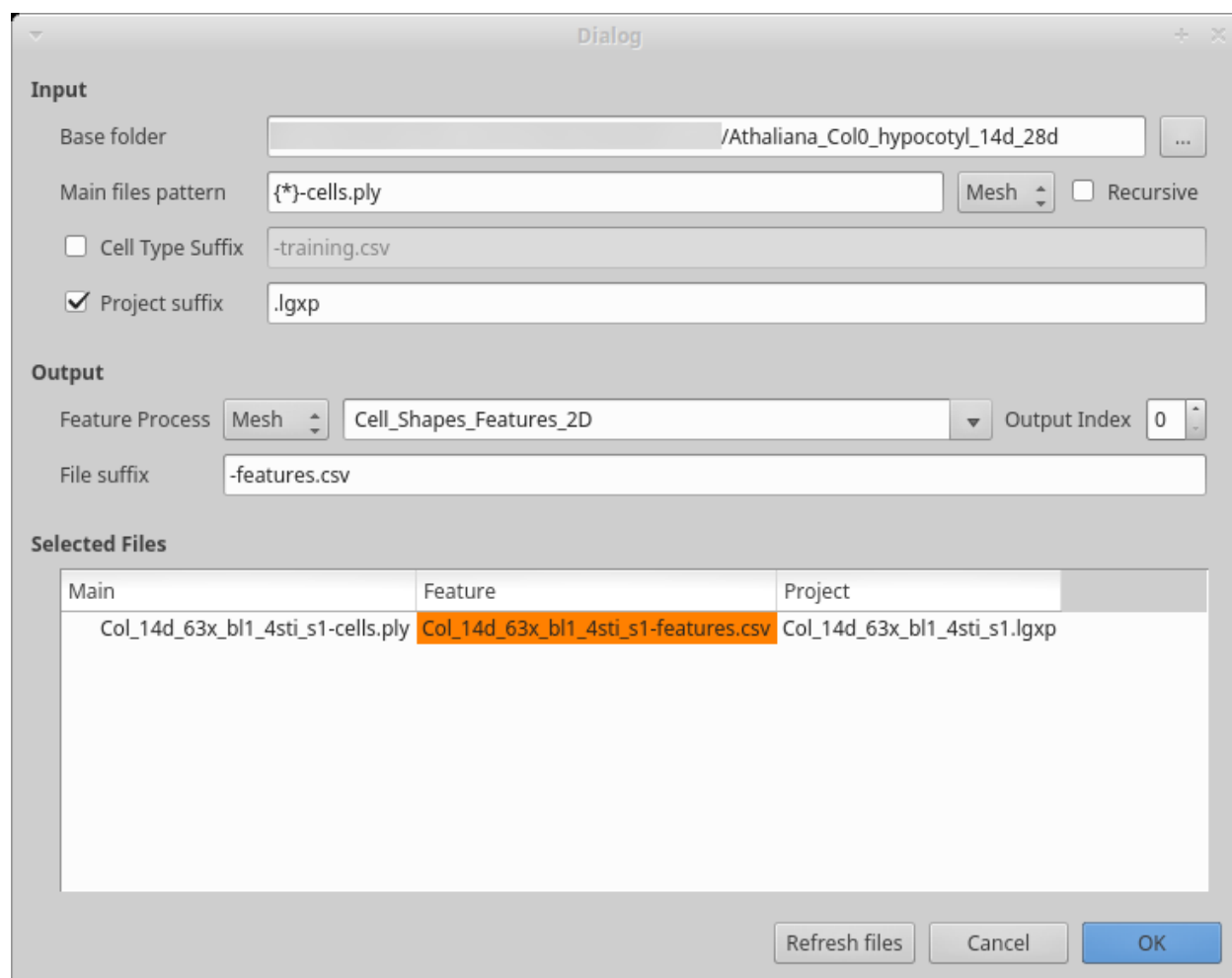
Features computation

Fig. 4.25: Dialog box to compute cells features.
The orange background indicates the feature file exists already and will be over-written.

1. Start the process creating the classifier features:

      Process `[Global]Cell Classifier/Generate Classifier Features`

2. If you have been editing all your files in the same folder, the base folder should be correct, otherwise, select the folder containing all your datasets. If you created separate folders for different datasets, check the `recursive` check box.

3. You may want to update the `Main file pattern` to reflect your convention. The base name is contained within the curly bracket and used as prefix for the other files. As you can see in the example, the mesh here also has a suffix.

4. Inspect the file list to make sure all the files will be properly processed as you want. Then press the `OK` button to start the process.

Training the classifier

After having computed the features on all your samples you can train the classifier.

1. Start the process creating the classifier itself:

   Process `[Global]Cell Classifier/Generate Cell Classifier`

2. The input area works like the previous dialog box. You will notice the process allows you to use only a range of cell types. This can be useful if you are not sure about detecting some cell types.

3. We currently offer two algorithms: Support Vector Machine (SVM) and Random Forest. Previous tests showed that SVM work better for this particular problem, but you can select more than one algorithm.

4. After running the algorithm, you should examine the summary generated. You should in particular look at the confusions matrix and report at the top the file:

```
Machine Learning Summary
========================
Selected classifier:  Support Vector Machine

Testing using set with 9 elements
---------------------------------

Confusion matrix:
Predicted  3   4   5   6   7   Sum
Real
3          2   0   0   0   0    2
4          0   1   0   0   1    2
5          0   0   0   0   0    0
6          0   0   1   0   0    1
7          0   0   0   0   4    4
Sum        2   1   1   0   5    9

Classification report:
           precision    recall   f1-score    support

        3       1.00      1.00      1.00          2
        4       1.00      0.50      0.67          2
        5       0.00      0.00      0.00          0
        6       0.00      0.00      0.00          1
        7       0.80      1.00      0.89          4

avg / total     0.80      0.78      0.77          9


Score:  77.7777777778 %

Algorithms Details
==================

Support Vector Machine
```
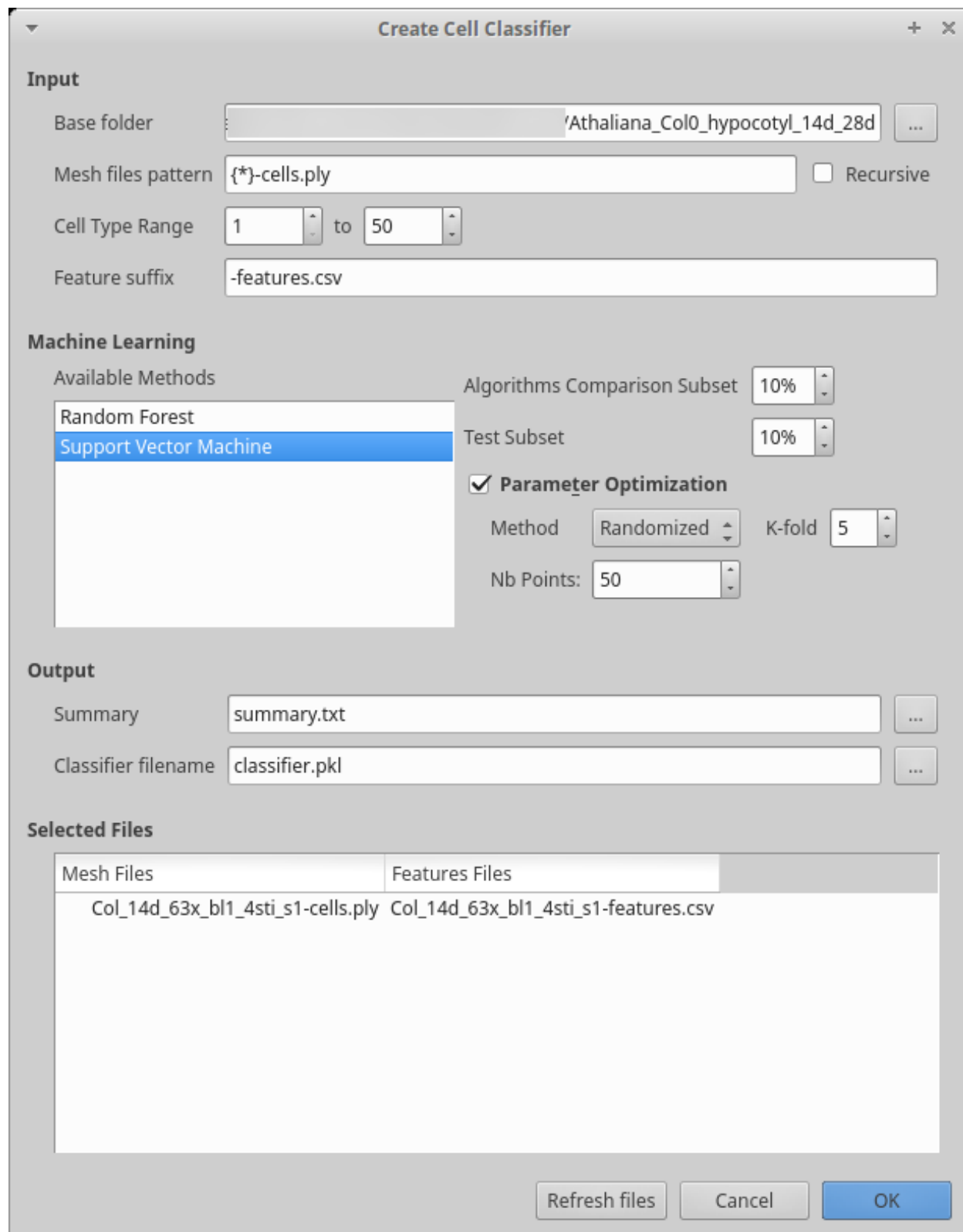
Fig. 4.26: Creation of the cell classifier.

```
----------------------

Best score with k-fold: 0.858823529412
Parameters:
class_weight : balanced
C : 31.04345899553238
gamma : 0.0514347955741465
```

Using the classifier
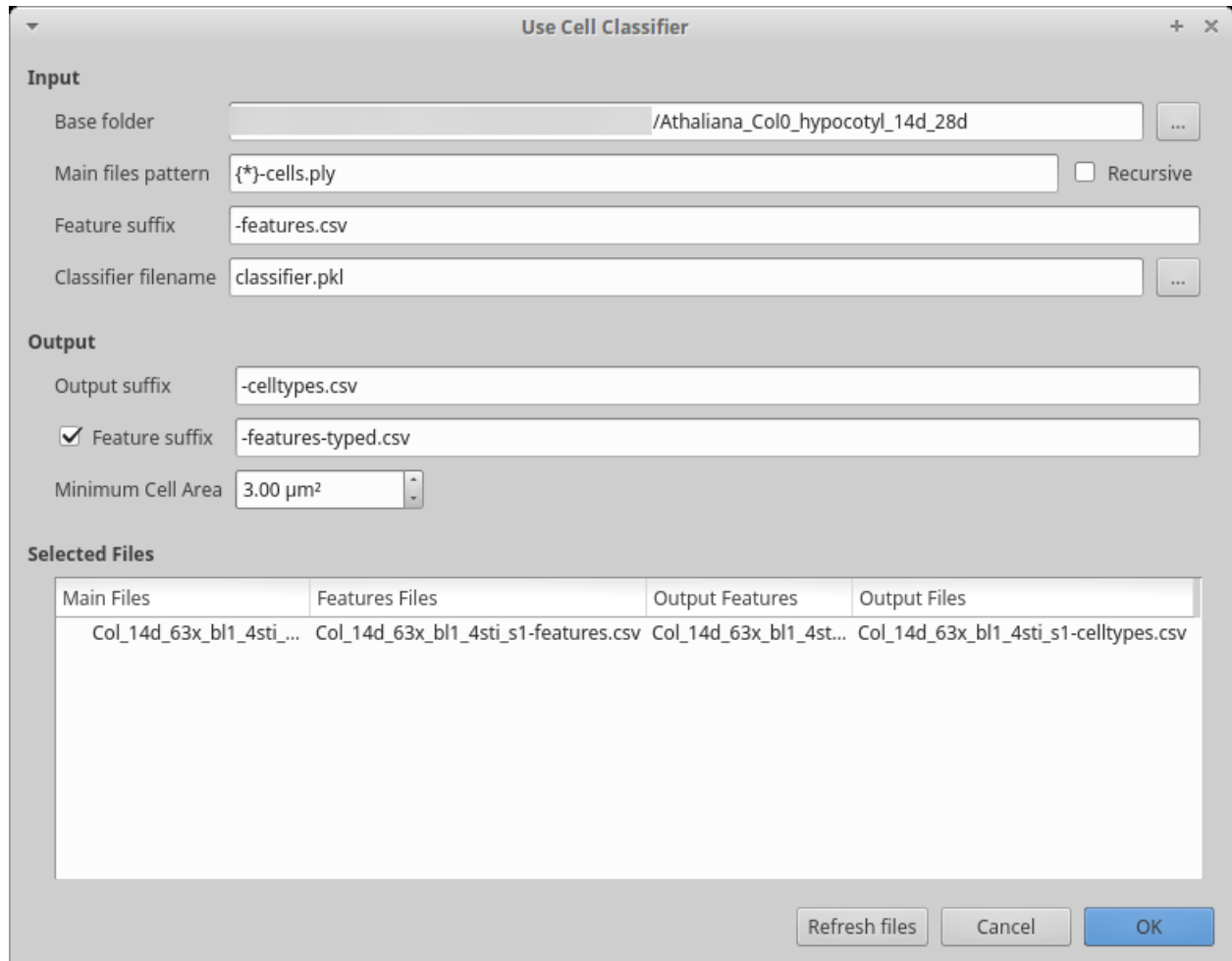
Fig. 4.27: Using of the cell classifier.

After creation, we want to use the cell classifier:

1. Start the process to use the classifier:

   Process   [Global]Cell Classifier/Use Cell Classifier

2. Choose the name of the classifier file you generated in the previous step

3. Fill in the input and output part as usual

4. In the output, the default is to write a CSV file containing two columns: the cell labels and the estimated cell type. You can optionally generate a file containing all the features and add a column with the estimated cell

type. This file can conveniently be used for further filtering/analyses using R, Python, ...

5. You can visualize the result by loading a mesh and changing the parents, using the files generated in the previous step (e.g. by default the ones ending up with -celltypes.csv)

> **Process**   `[Mesh]Lineage Tracking/Load Parents`

Fig. 4.28: Result of the cell classifier.

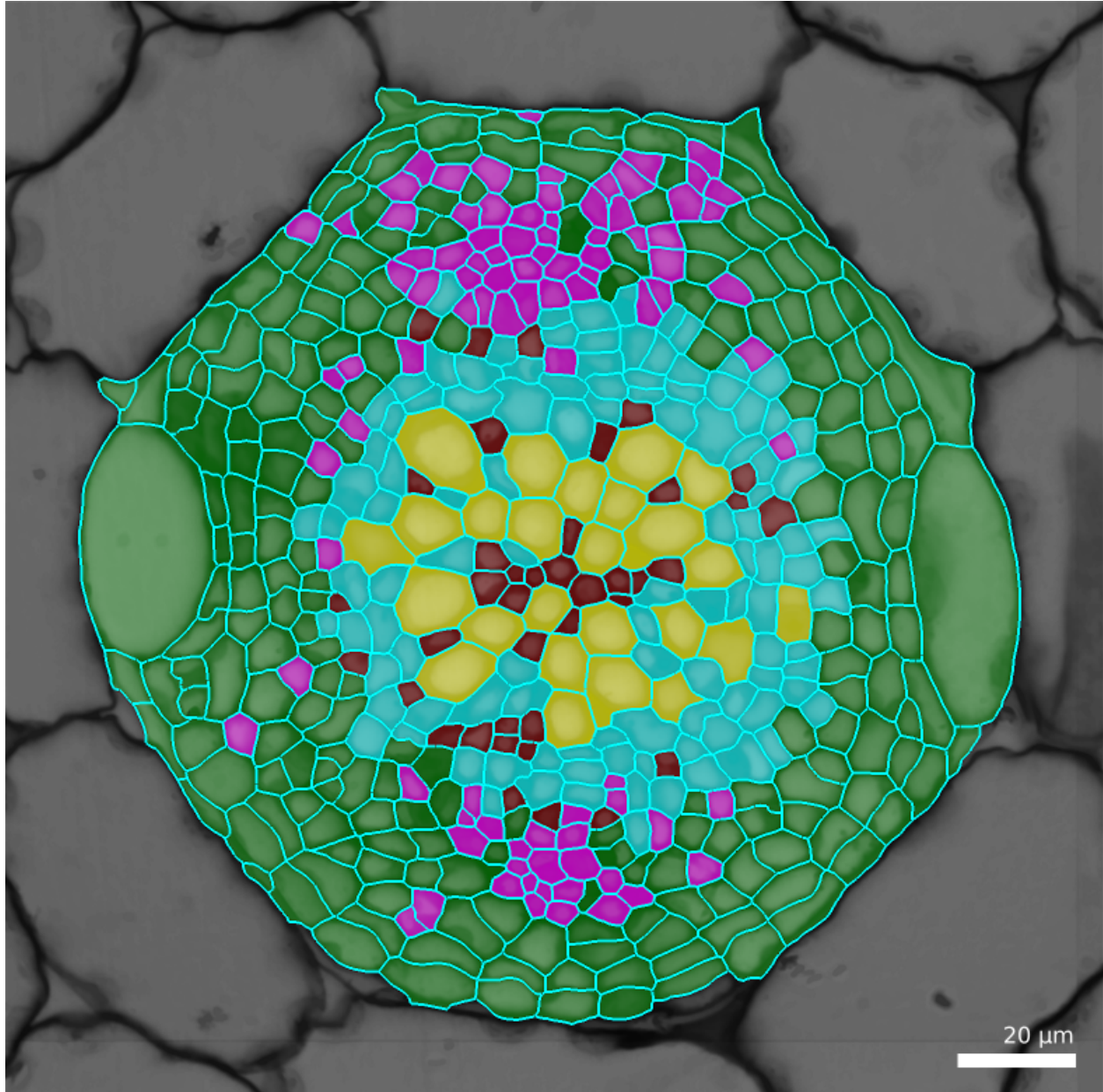## 4.7 Computing a growth map

TODO

Python Macros and Scripts

LithoGraphX can be scripted in python in two ways: scripts and macros.

In both cases, you can call any process with the syntax:

```
Type.Name(parameter1, parameter2, ...)
```

For example, to call the `Average` stack process, with a radius of 1 and 3 steps, you will type:

```
Stack.Average(1, 1, 1, 3)
```

The parameters are in the order in which they are presented, top to bottom. To make things easier, every time you run a process, the exact Python call is written in the file `LithoGraphX.py` in the current folder. The current folder is the one of the file marked in the title bar of the application.

In case of an error, three exceptions may be launched:

`UserCancelException`

> If the user canceled the process, this exception will be thrown.

`ProcessError`

> If an error occurs while the process runs, this exception is raised, and the attached string has the error message in it.

`NoSuchProcess`

> Exception thrown if the process doesn't exist.

5.1 Python Scripts

Python script are the simpler ones to make. Just write a script that will be executed using the `Global` process `Execute Python` in the `Python` folder.

5.2 Python Macros

Macros are more like processes. They are python modules, placed in the user or system macro folder. They appear at the same place as any other process. Also, while in scripts, the names `Stack`, `Mesh` and `Global` are always defined, in macros they need to be imported from the `lgxPy` module.

To be a valid macro, the module need to define a function `run`:

```python
def run(p1=default1, p2=default2, p3=default3, ...):
    ...
```

Remember that the parameter received are always strings, even if defaults are specified otherwise.

The number, name and default values of the parameters will be automatically extracted and presented in the user interface of LithoGraphX.

You can also define an `initialize` function, which need to take the same number of parameters as the `run` function, plus one. The extra parameter, placed at the end of the list, is the parent window, used to create dialog boxes. The function needs to return the list of parameters to pass to the run function:

```python
def initialize(p1, p2, p3, ..., parent):
    ...
    return (p1, p2, p3, ...)
```

Other variables that the module can defines are:

processName

> Name of the process, instead of the name of the module. If not specified, the default is the name of the module (e.g. the name of the file without the `.py`).

processType

> Must be one of `Stack`, `Mesh` or `Global`. If not `Global`, only processes of the same type can be called. If not specify, the default is `Global`.

processFolder

> Folder in which the process will be seen. If not specified, the default is `Python`.

parmsDescs

> Dictionnary giving, for each named parameter, a description. It will appear in the online parameter help.

parmsChoice

> Dictionnary giving, for each named parameter, a list of choices to display as drop-down list.

processIcon

> String with the path to the icon of the process. The path can be relative to the module path.

5.2.1 Example Macro

Here is a simple example macro, calling a number of times the blur process:

```python
# This first line is for modules to work with Python 2 or 3
from __future__ import print_function, division
# This is to access the Stack processes
from lgxPy import Stack
# This is to provide a user interface: note that for windows, LithoGraphX
# provides its own version of PyQt5
from PyQt5 import QtWidgets

processType = "Stack"
processName = "Repeat Blur"
processFolder = "Filters"
parmsChoice = dict(repeat=[1, 2, 3, 4, 5, 6, 7, 8, 9])
processIcon = ":/images/Blur.png" # we are re-using icons in the Qt resources
parmsDescs = dict(printed="Number of repetition of the blur")
```

```python
def initialize(X, Y, Z, repeat, parent):
    repeat, ok = QtWidgets.QInputDialog.getInt(parent, "Number of repeats", "N = ", int(repeat), 0, 1
    if not ok:
        return False
    return X, Y, Z, repeat


def run(X=.3, Y=.3, Z=.3, repeat=1):
    for i in range(int(repeat)):
        Stack.Gaussian_Blur_Stack(X, Y, Z)
```

Simply copy and paste this example in a file with a `py` extension and copy it to the user macro folder. The simplest way to access this folder is from the `Macros` menu in LithoGraphX. After that, select the `Reload Macros` entry in the same menu.

Hint: As LithoGraphX may be compiled with Python 2.7 or Pythong 3.4 or later, I recommend you use the `__future__` module.

Frequently Asked Questions

6.1 Installation

6.1.1 I have a Windows laptop with an NVidia discrete card, but the main windows doesn't show

You should have a terminal window opening. Check the start of the terminal, you should have something like this:

```
Welcome to LithoGraphX!

Thrust host evaluation: OpenMP

LithoGraphX version: 1.1.0 revision:
   created in thread: 0x14969344
Cuda driver version: 7.0
Cuda runtime version: 7.0

Cuda capable device found, device 0: GeForce GTX 960M
                Compute capability: 5.0
                      Total memory: 2047 Mb
                    MultiProcessors: 5
     Res.threads per MultiProcessor: 2048
             Max resident threads: 10240
                         Cuda cores: 960
                         Clock rate: 1.176 GHz
```

The version of the Cuda driver and runtime may be different, but you should have a valid version. If not, you probably need to install the NVidia drivers for your card, as often default video drivers are not compatible with Cuda. You will find the latest NVidia drivers here: http://www.nvidia.com/Download/index.aspx

6.1.2 I have a Windows laptop with an AMD discrete card, but the main windows doesn't show

You need to check if the discrete card has been triggered. For example, if you see something like:

```
OpenGL:
 - Version : 4.2.0 - Build 10.18.10.3540
 - Renderer: Intel(R) HD Graphics 4600
 - Vendor  : Intel
```

Then it's your onboard Intel card that is selected. And as of today (July 2015), no Intel card is capable of running LithoGraphX. You need to tell the drivers of your graphics card to use the discrete card.

6.1.3 I have a Linux laptop with an NVidia card and LithoGraphX doesn't start

Modern laptops with NVidia cards usually have a primary, low-consumption card. To use your NVidia card, you need to have either bumblebee or nvidia-prime installed (depending on how recent your Linux distribution is). If you installed nvidia-prime, make sure the nvidia card is selected. In a shell the command:

```
$ prime-select query
```

should return `nvidia`. If not, type:

```
$ sudo prime-select nvidia
```

If you installed bumblebee, you should have a program called optirun or primusrun (or both). LithoGraphX should detect automatically which one needs to be used. To make sure your installation is working run the following command in a shell:

```
$ primusrun glxinfo | grep -i nvidia
$ optirun glxinfo | grep -i nvidia
```

You should at least see:

```
server glx vendor string: NVIDIA Corporation
client glx vendor string: NVIDIA Corporation
```

If not, you need to fix your installation. You can try to contact us for help (http://www.lithographx.org/contact-credits).

6.1.4 I installed LithoGraphX from source and I get an error related to libgcc_s.so.1

The precise error should be:

```
libgcc_s.so.1 must be installed for pthread_cancel to work
```

We found this error can occur if you have g++ 4 and g++ 5 installed on your system (the minor version doesn't matter). The reason for this error is a change in the code generated by g++ 5 which leads to `libggc_s.so.1` being different. To solve this problem, make sure all libraries are either compiled with the same version of g++ than LithoGraphX, or at least that you use the compatibility flag for g++ 5 (see https://gcc.gnu.org/gcc-5/changes.html#libstdcxx). Note that we haven't tried compiling with the compatibility flag, so if you do this, please report on your success.

6.2 Referecing and citations

6.2.1 How to cite this software?

Please, cite this paper:

Barbier de Reuille, P., Routier-Kierzkowska, A.-L., Kierzkowski, D., Bassel, G.W., Schüp-bach, T., Tauriello, G., Bajpai, N., Strauss, S., Weber, A., Kiss, A., Burian, A., Hofhuis, H., Sapala, A., Lipowczan, M., Heimlicher M.B., Robinson, S., Bayer, E.M., Basler, K., Koumoutsakos, P., Roeder, A.H.K., Aegerter-Wilmsen, T., Nakayama, N., Tsiantis, M., Hay, A., Kwiatkowska, D., Xenarios, I., Kuhlemeier, C. & Smith, R.S. (2015) **MorphoGraphX: A platform for quantifying morphogenesis in 4D.** *eLife 4:e05864* http://dx.doi.org/10.7554/eLife.05864

Also, we would appreciate if you put a link to the LithoGraphX website http://lithographx.org in the publication, or even better, from your own website.

What's new?

7.1 Version 1.2.0

7.1.1 Highlights

- New Packaging system to distribute processes
- New Plugin system with interface to manager them
- Improved cell classification processes
- Optimization if ITK algorithms for 2D images
- New incremental auto-seeded watershed

7.1.2 Details

- Corrected reading RGB TIFFs and OME-TIFFs generated by the Zeiss software (Pull Request 1).
- Watershed can now be used on partially segmented stacks (Pull Request 2).
- Transfer function is now copied with the meta data of a store (Pull Request 3).
- Swap / Copy stacks handles small differences (e.g. less than noise) in step and origin correctly (Pull Request 3).
- Corrected cutting surfaces (Pull request 4).
- Improved Segment Section process: when inverting the image, it is reverted before exiting the process (Pull Request 5).
- Documented Shader class, also minor improvements of this class (Pull Request 5).
- Correction PC analysis: PCs are now ordered properly (Pull request 6)
- Split Ubuntu packages into `dev`, `doc` and `bin` and added versioning of shared object files (Pull Request 7).
- Now all ITK algorithms are instanciated explicitely for 2D images if one of the dimension is 1, it's faster and works more often (some 3D algorithms require a real 3D image) (Pull Request 8).
- Corrected the use of copyMetaData in ITK algorithms (Pull Request 8).
- Correctly compiles with newer version of ITK and CUDA (Pull Request 9).
- Improvements in cell classifier UI: the GUI has been fully revised and what was "suffixes" can now be Python's formatting expressions (Pull Request 10).

- Cell classifiers can now use project files to read the transformation before generating the features (Pull Request 10).

- The CSV reader now also understands if the CSV uses semi-colons instead of comas (looking at you Microsoft Excel) and it works if there is only one columns (Pull Request 11).

- The windows version now finds the correct version of Python (Pull Request 12).

- ITK segmentation now sets the labels flag properly (Pull Request 13).

- New section about hardware specification in the documentation (Pull Request 14).

- Corrected computation of the incline angle for 2d cell features to be in the range from -pi/2 to pi/2 (Pull Request 15).

- The cell classifier has been moved to Global processes to allow use of cells from stacks or meshes and change the transformation (Pull Request 16).

- Added packaging system and plugin management (Pull Request 18 and Pull Request 19).

- For windows, added a building environment (Pull Request 18).

- Added new tutorial on "flat" cells segmentation (Pull Request 18).

- Corrected problem with drawing when selecting cells (Pull Request 18).

- Corrected rendering of Pixel Edit's circle (Pull Request 19).

- Added dialog box to use laptop or desktop configuration the first time LithoGraphX is launched (Pull Request 19).

- Updated tutorials with new presentation of used processes (Pull Request 19).

- Thrust can now compile in CPU mode, in which case CLang can be used (Pull Request 19).

- Under windows, the installer register LithoGraphX to the local installation of cmake (Pull Request 19).

- All tools now have keyboard shortcuts (Pull Request 19).

- Under windows, the installer now offers to also install Image Magick (Pull Request 19).

- Add new "What's new?" section in the documentation (Pull Request 20).

- When a new version of LithoGraphX appears, the About box appears before the interface (Pull Request 20).

Contributors and Licensing

LithoGraphX is licensed under the GPL v3. It is a fork of MorphoGraphX version 1.0 rev. 1256.

8.1 Contributors

- Dr. Barbier de Reuille, Pierre (lead developer)
- Dr. Robinson, Sarah
- Dr. Burian, Agata
- Dr. Summers, Holly
- Dr. Yoshida, Saiko

8.2 Past Contributors

People listed have contributed to MorphoGraphX in significant ways, and in this way also contributed to LithoGraphX:

- Dr. Weber, Alain
- Schuepbach, Thierry
- Toriello, Gerardo
- Dr. Strauss, Soeren
- Dr. Nakayama, Naomi
- Dr. Richard Smith's team

8.3 Funding Support

- SystemsX.ch
- Swiss National Science Foundation
- University of Bern, Switzerland

Indices and tables

- genindex
- modindex
- search

[Yoshida2014] Yoshida, S., Barbier de Reuille, P., Lane, B., Bassel, G. W., Prusinkiewicz, P., Smith, R. S. and Weijers, D. **Genetic Control of Plant Development by Overriding a Geometric Division Rule** *Developmental Cell*, 2014, 29, 75-87

[Boudaoud2014] Boudaoud, A.; Burian, A.; Borowska-Wykręt, D.; Uyttewaal, M.; Wrzalik, R.; Kwiatkowska, D. and Hamant, O. **FibrilTool, an ImageJ plug-in to quantify fibrillar structures in raw microscopy images**. *Nature Protocols*, 2014, 9, 457-463

[BarbierDeReuille.Ragni.InPress] Barbier de Reuille P, Ragni L. in press. Vascular morphodynamics during secondary growth. In: de Lucas M, Etchells JP, eds. *Xylem: methods and protocols* Springer.