

# ResolveRT™ 4.0

User's and Reference Guides

---

amira for microscopy  
including Skeleton Pack

## Copyright Information

©1995-2005 Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Germany

©1999-2005 Mercury Computer Systems.

All rights reserved.

### Trademark Information:

amira and ResolveRT are being jointly developed by Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Mercury Computer Systems.

amira ® is a registered trademark of Konrad-Zuse-Zentrum für Informationstechnik Berlin.

ResolveRT <sup>TM</sup> is a trademark of Mercury Computer Systems.

HardCopy, MeshViz, VolumeViz, TerrainViz are trademarks of Mercury Computer Systems S.A.

Mercury Computer Systems S.A. is a source licensee of OpenGL®, Open Inventor® from Silicon Graphics, Inc. OpenGL® and Open Inventor® are registered trademarks of Silicon Graphics, Inc.

All other products and company names are trademarks or registered trademarks of their respective companies.

This manual has been prepared for Mercury Computer Systems licensees solely for use in connection with software supplied by Mercury Computer Systems and is furnished under a written license agreement. This material may not be used, reproduced or disclosed, in whole or in part, except as permitted in the license agreement or by prior written authorization of Mercury Computer Systems. Users are cautioned that Mercury Computer Systems reserves the right to make changes without notice to the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical, arithmetic or listing errors.

# Contents

|          |                                                                 |           |
|----------|-----------------------------------------------------------------|-----------|
| <b>I</b> | <b>ResolveRT User's Guide(amira for microscopy)</b>             | <b>1</b>  |
| <b>1</b> | <b>Deconvolution introduction</b>                               | <b>3</b>  |
| 1.1      | General remarks about image deconvolution . . . . .             | 4         |
| 1.2      | Data acquisition and sampling rates . . . . .                   | 5         |
| 1.3      | Standard Deconvolution Tutorial . . . . .                       | 6         |
| 1.4      | Blind Deconvolution Tutorial . . . . .                          | 11        |
| 1.5      | Bead Extraction Tutorial . . . . .                              | 14        |
| 1.6      | Performance issues and multi-processing . . . . .               | 18        |
| <b>2</b> | <b>Working with Multi-Channel Images</b>                        | <b>21</b> |
| 2.1      | Loading Multi-Channel Images into amira . . . . .               | 21        |
| 2.2      | Using OrthoSlice with a MultiChannelField . . . . .             | 22        |
| 2.3      | Using ProjectionView with a MultiChannelField . . . . .         | 23        |
| 2.4      | Using Voltex with a MultiChannelField . . . . .                 | 23        |
| 2.5      | Saving a MultiChannelField in a Single AmiraMesh File . . . . . | 25        |
| <b>3</b> | <b>Skeleton Tutorial</b>                                        | <b>27</b> |
| 3.1      | Importing your Image Data . . . . .                             | 27        |
| 3.2      | Arranging the Bricks . . . . .                                  | 28        |
| 3.3      | Aligning Bricks . . . . .                                       | 29        |
| 3.4      | Filtering, Correcting Z-Drop, Resampling . . . . .              | 29        |
| 3.5      | Creating the Large Disk Data . . . . .                          | 30        |

|      |                                                           |    |
|------|-----------------------------------------------------------|----|
| 3.6  | Accessing the Large Disk Data . . . . .                   | 31 |
| 3.7  | Computing directly on the Large Disk Data . . . . .       | 32 |
| 3.8  | Region of Interest . . . . .                              | 34 |
| 3.9  | Check Network . . . . .                                   | 36 |
| 3.10 | Coloring a Lineset According to its Depth Value . . . . . | 36 |

## **II Reference Guide - Alphabetic Index of Modules** **37**

### **4 ResolveRT** **39**

|     |                            |    |
|-----|----------------------------|----|
| 4.1 | BeadExtract . . . . .      | 39 |
| 4.2 | Convolution . . . . .      | 40 |
| 4.3 | CorrectZDrop . . . . .     | 41 |
| 4.4 | DataPreprocess . . . . .   | 42 |
| 4.5 | Deconvolution . . . . .    | 43 |
| 4.6 | DistanceMap . . . . .      | 45 |
| 4.7 | FourierTransform . . . . . | 47 |
| 4.8 | PSFGen . . . . .           | 48 |

### **5 Skeleton Pack** **49**

|      |                                 |    |
|------|---------------------------------|----|
| 5.1  | AlignBlocks . . . . .           | 49 |
| 5.2  | ApplyMask . . . . .             | 50 |
| 5.3  | ApplyTemplateToMosaic . . . . . | 51 |
| 5.4  | ChamferMap . . . . .            | 53 |
| 5.5  | CheckNetwork . . . . .          | 54 |
| 5.6  | DisplayMosaic . . . . .         | 56 |
| 5.7  | EvalOnLines . . . . .           | 56 |
| 5.8  | MosaicToLargeDiskData . . . . . | 56 |
| 5.9  | Thinner . . . . .               | 58 |
| 5.10 | Threshold . . . . .             | 58 |
| 5.11 | TraceLines . . . . .            | 59 |

|            |                                                           |           |
|------------|-----------------------------------------------------------|-----------|
| <b>III</b> | <b>Reference Guide - Alphabetic Index of File Formats</b> | <b>61</b> |
| <b>6</b>   | <b>ResolveRT</b>                                          | <b>63</b> |
| 6.1        | Analyze 7.5 . . . . .                                     | 63        |
| 6.2        | AnalyzeAVW . . . . .                                      | 63        |
| 6.3        | Bio-Rad Confocal Format . . . . .                         | 64        |
| 6.4        | Leica 3D TIFF . . . . .                                   | 64        |
| 6.5        | Leica Binary Format (.lei) . . . . .                      | 64        |
| 6.6        | Leica Slice Series (.info) . . . . .                      | 64        |
| 6.7        | MRC . . . . .                                             | 65        |
| 6.8        | Metamorph STK Format . . . . .                            | 65        |
| 6.9        | Stacked-Slices . . . . .                                  | 65        |



## **Part I**

# **ResolveRT User's Guide**

**(amira for microscopy)**



# Chapter 1

## Deconvolution introduction

ResolveRT's deconvolution provides powerful algorithms for improving the quality of microscopic images recorded by 3D widefield and confocal microscopes. Two different methods are supported, namely a so-called non-blind and a blind deconvolution method, both based on iterative maximum-likelihood image restoration. In the first case a measured or computed point spread function (PSF) is required. In the second case the PSF is estimated along with the data itself.

The deconvolution documentation is organized as follows:

- *General remarks about image deconvolution*
- *Data acquisition and sampling rates*
- *Standard deconvolution tutorial*
- *Blind deconvolution tutorial*
- *Bead extraction tutorial*
- *Performance issues and multi-processing*

The following modules are included:

- *BeadExtract* - obtain a PSF from a bead measurement
- *Convolution* - convolve two 3D images
- *CorrectZDrop* - corrects attenuation in z-direction
- *DataPreprocess* - background and flatfield correction
- *Deconvolution* - the actual deconvolution front-end
- *FourierTransform* - computes FFT and power spectrum
- *PSFGenerate* - calculates a theoretical PSF

## 1.1 General remarks about image deconvolution

Deconvolution is a technique for removing out-of-focus light in a series of images recorded via optical sectioning microscopy. Intended to investigate 3D biological objects, optical sectioning microscopy works by creating multiple images (optical sections) of a fluorescing object, each with a different focus plane. However, besides the in-focus structures, the images usually also contain out-of-focus light from other parts of the object, causing haze and severe axial blur. This is even the case for a confocal laser scanning microscope, where most of the out-of-focus light is removed from the image by a pinhole system. Mathematically, the image produced by any microscopic system can be described as the convolution of the ideal unblurred image of the specimen and the microscope's so-called point spread function (PSF), i.e., the image of an ideal point light source. With the inverse of this process, called deconvolution, a deblurred image of the specimen can be obtained, provided the point spread function is known, or at least can be estimated.

The amira deconvolution modules mainly provide two variants of a powerful iterative maximum-likelihood image restoration algorithm, namely a non-blind one and a blind one. The difference between them is that in the first case a measured or computed point spread function is used, while in the second case the PSF is estimated along with the data itself. Maximum-likelihood image restoration can be considered as the de-facto standard for deconvolution of 3D optical sections. Although computationally quite expensive, the method is able to significantly enhance image quality. At the same time it is very robust and insensitive with respect to noise artifacts. However, it should be noted that, although rejecting most of the out-of-focus light, by no means all of it is rejected. Therefore, some noticeable haze remains in the images. Also, the images retain a substantial axial smearing in z-direction, which cannot be removed by any deconvolution algorithm.

At first sight, one may wonder why both a non-blind and a blind deconvolution algorithm are provided; blind deconvolution seems to be more general because the PSF is calculated automatically. One answer is that blind deconvolution is computationally even more expensive than non-blind iterative maximum-likelihood image restoration. The other answer is that in a blind deconvolution algorithm a meaningful estimate of the PSF can only be computed if severe constraints are imposed. For example, a trivial solution of the blind deconvolution problem would be an image which is identical to the input image and a PSF with the shape of an ideal delta peak. Obviously, this solution isn't useful at all. Therefore, if for example confocal data is to be deconvolved, the algorithm fits the actual PSF in such a way that it looks like a possible measured PSF of a confocal microscope. More precisely, the fit is constrained to be in agreement with the experimental parameters (the refractive index of the medium, the numerical aperture of the objective, and the voxel sizes). Sometimes this can lead to wrong results, for example when the confocal pinhole aperture of the microscope wasn't stopped down sufficiently during confocal image acquisition, in which case the microscope actually didn't behave like a true confocal microscope. As a matter of fact you should try which approach provides the best results for your own image data, blind deconvolution or non-blind deconvolution with either a measured or an automatically computed PSF.

## 1.2 Data acquisition and sampling rates

In order to obtain best quality when deconvolving microscopic images some fundamental guidelines should be obeyed during image acquisition. Good results may be obtained even if some of these guidelines are not followed exactly, but in general the chances to get satisfactory results improve if they are. Below we discuss the most important recommendations.

### Adjusting the scanned image volume

The region of interest should be centered in the middle of the image volume, as the optics of the microscope has usually the least aberrations in this region and it helps to avoid possible boundary artifacts, which can arise during the deconvolution procedure. Especially for widefield data it is important to record a sufficiently large (preferably empty) region below and above the actual sample. Ideally, this region should be as large as the sample itself. For example, if the sample covers 100 micrometers in the z-direction, the scanned image volume should range from 50 micrometers below the sample to 50 micrometers above it.

### Choosing the right sampling rate

The sampling rate is determined by the pixel sizes in the x and y directions as well as the distance between two subsequent optical sections, both measured in micrometers. Generally speaking, image deconvolution works best if the data is apparently oversampled, i.e., if the pixel or optical section spacing is smaller than required. The maximal required sampling distance (Nyquist sampling) to avoid ambiguities in the data can be obtained from considerations in Fourier-space yielding

$$d_{xy} = \frac{\lambda}{4 NA},$$

where  $\lambda$  denotes the wavelength and  $NA$  is the numerical aperture of the microscope. Similar considerations yield for the maximal distance between adjacent image planes:

$$d_z = \frac{\lambda}{2 n (1 - \cos(\alpha))},$$

where  $n$  denotes the refractive index of the object medium and  $\alpha$  the aperture half angle as determined by  $NA = n \sin(\alpha)$ .

For a confocal microscope, both the in-plane sampling distance and the axial sampling distance need to be in theory approximately 2 times smaller. However, this requirement is far too strict for most practical cases and even in the widefield case, approximately fulfilling the above requirements is often sufficient.

The total number of optical sections is obtained by dividing the height of the image volume by the sampling distance  $d_z$ . It should be mentioned that deconvolution also works if the sampling distances are

not matched rigorously, but matching them improves the chances to get good results. In general, over-sampling the object is less harmful than undersampling it, with one exception: In the case of confocal data, the sampling distance  $d_{xy}$  should not be much smaller than indicated, if the blind deconvolution algorithm or the non-blind deconvolution algorithm together with a theoretically computed confocal PSF are used. Otherwise the unconstrained Maximum Likelihood algorithm and the predominant noise in the data might lead to unsatisfactory results.

## Black level and saturation

Before grabbing images from the microscope's camera, the light level should be adjusted in such a way that saturated pixels, either black or white ones, are avoided. Saturated pixels are pixels which are clamped to either black or white because their actual intensity values are outside the range of representable intensities. In any case, saturation means a loss of information and thus prevents proper post-processing or deconvolution. At the same time, a high background level should be avoided because it decreases the dynamic range of the imaging system and the deconvolution works worse. This means that empty regions not showing any fluorescence should appear almost black. A background level close to zero is especially important when bead measurements are performed in order to extract an experimental point spread function. Details are discussed in a separate tutorial about *bead extraction*.

## 1.3 Standard Deconvolution Tutorial

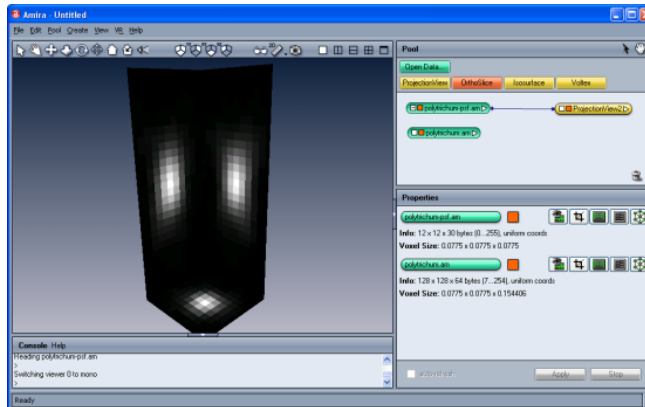
This tutorial explains how 3D image data sets can be deconvolved in *amira*. It is assumed that the reader is already familiar with the basic concepts of *amira* itself. If this is not the case, it is strongly recommended to work through the *standard amira tutorials* first. In this section the following topics are covered:

1. Prerequisites for deconvolution
2. Resampling a measured PSF
3. Deconvolving an image data set
4. Calculating a theoretical PSF

As an example we are going to use a confocal test data set (*polytrichum.am*) provided with the *amira* deconvolution modules. The data file is located in the directory *Amira-4.0/data/deconv*.

- Load the data set *polytrichum.am*.
- Visualize it, for example, using a *ProjectionView* module.

The data set shows four chloroplasts in a spore of the moss *polytrichum commune*.



**Figure 1.1:** Maximum intensity projection of *polytrichum-psf.am*

## Prerequisites for deconvolution

Besides the image data itself, for the standard non-blind deconvolution algorithm a so-called *point spread function* (PSF) is also required. The PSF is the image of a single point source, or as a close approximation, the image of a single fluorescing sub-resolution sphere. PSF images can either be computed from theory (see below) or they can be obtained from measurements. In the latter case tiny so-called *beads* are recorded under the same conditions as the actual object. This means that the same objective lens, the same dye and wavelength, and the same immersion medium are used. Typically, the images of multiple beads are averaged to obtain an estimate of a single PSF. *amira* provides a module called *BeadExtract* facilitating this process. The use of this module is discussed in a *separate tutorial* about bead extraction. At this point let us simply load a measured PSF from a file.

- Load the data set *polytrichum-psf.am*.
- Use the *ProjectionView* module to visualize it.

The PSF appears as a bright spot located in the middle of the image volume (Figure 1.1). It is important that the PSF is exactly centered. Otherwise, the deconvolved data set will be shifted with respect to the original image. Also, it is important that the PSF fades out to black at the boundaries. If this is not the case, the black level of the PSF image needs to be adjusted using the *Arithmetic* module. Finally, neither the PSF nor the image to be deconvolved should exhibit *intensity attenuation artifacts*, i.e., image slices with decreased average intensity due to excessive light absorption in other slices. If such artifacts are present, they can be removed using the *CorrectZDrop* module.



Figure 1.2: Resampling a PSF using the *Resample* module.

## Resampling a measured PSF

Next, select both, the PSF and the image data. You'll notice that the voxel sizes of both objects are not the same. It is recommended to adjust different voxel sizes of PSF and image data prior to deconvolution using the *Resample* module. The deconvolution module itself also accounts for different voxel sizes, but it does so by using point sampling with trilinear interpolation. This is OK as long as the voxel size of the PSF is larger than that of the image data. However, in our case the voxel size of the PSF is smaller than that of the image data, i.e., the resolution of the PSF higher. Using the *Resample* module provides slightly more accurate results here, since all samples will be filtered correctly using a Lanczos kernel.

- Connect a *Resample* module to *polytrichum-psf.am*.
- Connect the *Reference* port of the *Resample* module to the image data set *polytrichum.am*
- In the *Mode* port of the *Resample* module, choose *voxel size* (see Figure 1.2).
- Resample the PSF by pressing the *Apply* button.

The *voxel size* option means that the PSF will be resampled on a grid with exactly the same voxel size as the image data set, which is connected to the *Reference* port. While the original PSF had a resolution of 12 x 12 x 30 voxels, the resampled one only has 12 x 12 x 16 voxels. However, the extent of a single voxel in z-direction is bigger now.

## Deconvolving an image data set

After a suitable PSF has been obtained we are ready for deconvolving the image data set. This can be done by attaching a *Deconvolution* module to the image data.

- Connect a *Deconvolution* module to *polytrichum.am*.
- Connect the *Kernel* port of the *Deconvolution* module to the resampled PSF *polytrichum-psf.Resampled*.

Once the deconvolution module is connected to its two input objects, some additional parameters need to be adjusted (for a detailed discussion of these parameters see also the *reference documentation* of the *Deconvolution* module itself). Figure 1.3 shows these settings:

**Border width:** For deconvolution the image data has to be enlarged by a guardband region. Otherwise boundary artifacts can occur, i.e., information from one side of the data can be passed to the other. There is no need to make the border bigger than the size of the PSF. However, if the data set is dark at the boundaries, a smaller border width is sufficient. In our case, let us choose the border values 0, 0, and 8 in the x, y, and z direction.

**Iterations:** The number of iterations of the deconvolution algorithm. Let us choose a value of 20 here.

**Initial estimate:** Specifies the initial estimate of the deconvolution algorithm. If *const* is chosen a constant image is used initially. This is the most robust choice, yielding good results even if the input data is very noisy. We keep this option here.

**Overrelaxation:** Overrelaxation is a technique to speed up the convergence of the iterative deconvolution process. In most cases the best compromise between speed and quality is *fixed* overrelaxation. Therefore we keep this choice also.

**Method:** Selects between standard (non-blind) and blind deconvolution. Let us specify the *standard* option here.

The actual deconvolution process is started by pressing the *Apply* button. Please press this button now. The deconvolution should take about 1-2 minutes on a modern computer. During the deconvolution the progress bar informs you about the status of the operation. Also, after every iteration a message is printed in the *amira* console window indicating the amount of change of the data. If the change seems to be small enough, you can terminate the deconvolution procedure by pressing the *Stop* button. However, note that the stop button is evaluated only once between two consecutive iterations.

When deconvolution is finished, a new data set called *polytrichum.deconv* appears in the Pool. You might take a look at the deconvolved data by moving the *ProjectionView* connection line from *polytrichum.am* to *polytrichum.deconv*.

## Calculating a theoretical PSF

Sometimes bead measurements are difficult to perform, so that an experimental PSF cannot easily be obtained. In such cases a theoretical PSF can be used instead. *amira* provides the module *PSFGen*,

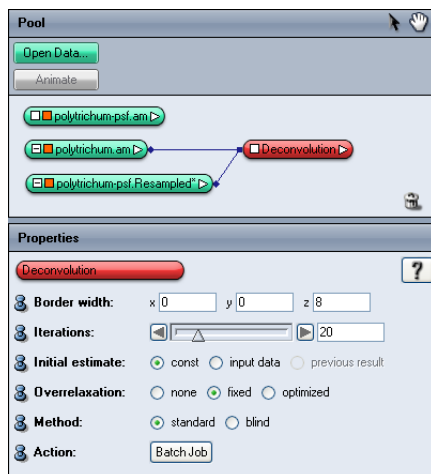


Figure 1.3: Deconvolution module attached to *polytrichum.am*.

allowing you to calculate theoretical PSFs. The module can be created by selecting *PSFGen* from the *Create Others* menu of the amira main window.

Once the module is created again some parameters have to be entered. The *resolution* and the *voxel size* can be most easily specified by connecting the *Data* port of the *PSFGen* module to the image data set to be convolved. In our case, please connect this port to *polytrichum.am*.

In order to generate a PSF, you also need to know the numerical aperture of the microscope objective, the wavelength of the emitted light (to be entered in micrometers!), and the refractive index of the immersion medium. In our test example these values are NA=1.4, lambda=0.58, and n=1.516 (oil medium). Also, change the microscopic mode from widefield to confocal.

After you press the *Apply* button, the computed PSF appears as an icon labelled *PSF* in the Pool. You can compare the theoretical PSF with the measured one using the *OrthoSlice* module. You'll notice that the measured PSF appears to be slightly wider. This is a common observation in many experiments.

Once you have computed a theoretical PSF, you can perform non-blind deconvolution as described above. However, for convenience the *Deconvolution* module is also able to compute a theoretical PSF by itself. You can check this by disconnecting the *Kernel* port of the *Deconvolution* module. If no input is present at this port, additional input fields are shown, allowing you to enter the same parameters (numerical aperture, wavelength, refractive index, and microscopic mode) as in *PSFGen*. After these parameters have been entered, the deconvolution process again can be started by pressing the *Apply* button.

Note that any previous result connected to the *Deconvolution* module will be overwritten when starting the deconvolution process again. Therefore, be sure to disconnect a previous result if you want to compare deconvolution with different input PSFs.

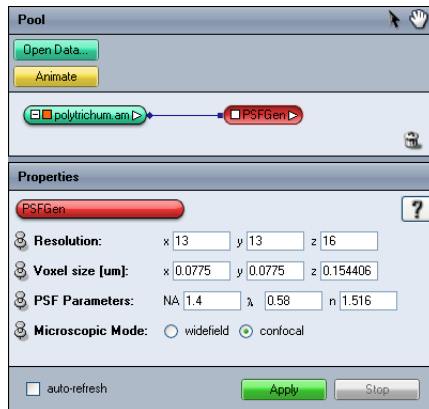


Figure 1.4: The PSFGen module calculates theoretical PSFs.

## 1.4 Blind Deconvolution Tutorial

This tutorial explains how blind deconvolution can be performed in *amira*. At the same time it describes how deconvolution jobs can be processed using the *amira* job queue. Like in the previous tutorial, it is assumed that the reader is already familiar with the basic concepts of *amira* itself. If not, we recommend to work through the *standard amira tutorials* first.

### A blind deconvolution example

Let us start by loading a raw image data set first.

- Load the file *alphalobe.am* from the directory *Amira-4.0/data/deconv*.
- Visualize the data set by attaching a *ProjectionView* module to it.

The data set has been recorded using a standard fluorescence microscope under so-called *widefield* conditions. It shows a neuron from the alpha-lobe of the honeybee brain. Compared to the confocal data set used in the standard deconvolution tutorial, *alphalobe.am* is much bigger. It has a resolution of 248 x 248 x 256 voxels with a uniform voxel size of 1 micrometer. In the xy-plane of the projection view the structure of the neuron can be clearly identified. However, the contrast of the image is quite poor because there is a significant amount of out-of-focus light or haze present. With *amira*'s blind deconvolution algorithm we can enhance the image data without needing to know an explicit PSF in advance.

- Attach a deconvolution module to *alphalobe.am*.
- Adjust the parameters like shown in Figure 1.5.

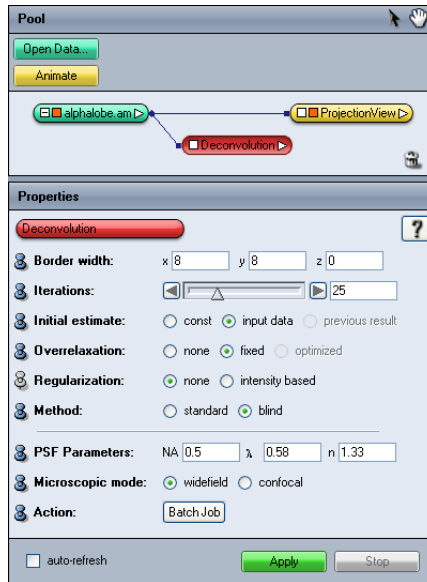


Figure 1.5: Parameters for blind deconvolution.

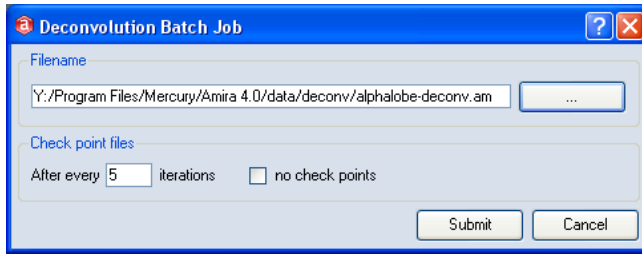
The individual parameters have the following meaning:

**Border width:** Like for standard non-blind deconvolution, the image data has to be enlarged by a guardband region. Otherwise boundary artifacts can occur, i.e., information from one side of the data can be passed to the other. In our case we only provide a small guardband region of 8 voxels in x- and y-direction. In z-direction we do not provide any border because there are sufficiently many empty slices below and above the actual neuron. The resulting size of the data arrays on which the computations are performed then is  $256 \times 256 \times 256$ . Because 256 is a power of two ( $2^8$ ), the Fast Fourier Transforms, the computationally most expensive part of the deconvolution algorithm, can be executed somewhat faster.

**Iterations:** We choose a value of 25 here. Depending on the data, usually at least 10 iterations are required. With overrelaxation being enabled (see below), results usually don't improve much after 40 iterations.

**Initial estimate:** Specifies the initial estimate of the deconvolution algorithm. Since there is not much noise present in the original alphalobe images it is safe to choose *input data* here. This causes the algorithm to converge even faster.

**Overrelaxation:** Overrelaxation is a technique to speed up the convergence of the iterative deconvolution process. We enable overrelaxation by choosing the *fixed* toggle. **Regularization:** We chose *none* here in order to do no regularization.



**Figure 1.6:** Dialog for submitting a deconvolution job.

*Method:* We chose *blind* here in order to select the blind deconvolution algorithm.

*PSF Parameters:* For *alphalobe.am* the numerical aperture is 0.5, the wavelength is 0.58 micrometers, and the refractive index is 1.33 (water). These parameters are required in order to apply certain constraints to the estimated point spread function. They are also used in order to compute an initial PSF. If a data set would be connected to the *Kernel* port of the deconvolution module, this data set would be used as the initial PSF with the given PSF parameters still acting as constraints. For example, you could provide a measured PSF and let it be fitted to the actual data by the deconvolution algorithm.

*Microscopic mode:* *alphalobe.am* is a widefield data set, so select this option here.

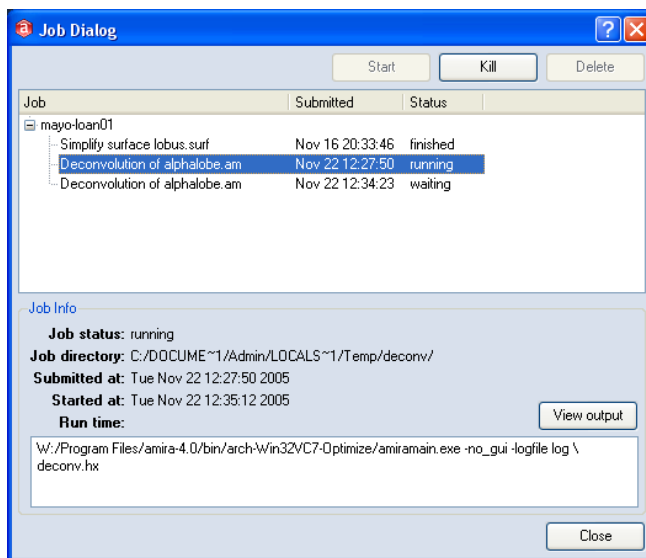
## Submitting a deconvolution job

After all parameters have been entered, the deconvolution process can be started. On a modern computer, blind deconvolution of our test data set roughly takes about 20 minutes. Especially, if you want to deconvolve multiple data sets at once it is inconvenient to do this in an interactive session. Therefore multiple deconvolution jobs can be submitted to the *amira* job queue and then, for example, processed overnight. This works as follows:

- Press the *Batch Job* button of the *Action* port. A dialog as shown in Figure 1.6 pops up.
- In the dialog choose a file name under which you want to save the deconvolved data set, e.g. `C:/Temp/alphalobe-deconv.am`.
- Modify the text field, so that check point files are written after every 5 iterations.

Check point files are used to store intermediate results. With the above settings the deconvolved data is written into a file after every 5 iterations. Check point files are named like the final result, but a consecutive number is inserted just before the file name suffix. For example, if the result file name is `C:/Temp/alphalobe-deconv.am`, the check point files are named `C:/Temp/alphalobe-deconv-0005.am`, `C:/Temp/alphalobe-deconv-0010.am` and so on. Now we are ready to actually submit the batch job.

- Press the *Submit* button of the deconvolution dialog. After a few seconds the *amira* batch job



**Figure 1.7:** The amira job dialog showing a pending deconvolution job.

dialog appears, compare Figure 1.7.

- Select the deconvolution job and press the *Start* button.

You now have to wait about 20 minutes until the deconvolution job is finished. Once the job queue has been started, you can quit amira. The batch jobs will be continued automatically. If amira is still running when the deconvolution job exits then the result will be loaded automatically in amira. Otherwise you have to restart amira and load the deconvolved data set manually.

## 1.5 Bead Extraction Tutorial

Non-blind deconvolution is a powerful and robust method for enhancing the quality of 3D microscopic images. However, the method requires that the image of the point spread function (PSF) responsible for image blurring is provided. As stated in the *standard deconvolution tutorial*, the PSF can either be calculated theoretically or it can be obtained from a bead measurement. amira provides a special-purpose module called *BeadExtract* which facilitates the extraction of PSF images from one or multiple bead measurements. In this tutorial the use of the module shall be explained. The following topics are covered:

1. Bead measurements
2. Projection View and Projection View Cursor

### 3. Resampling and averaging the beads

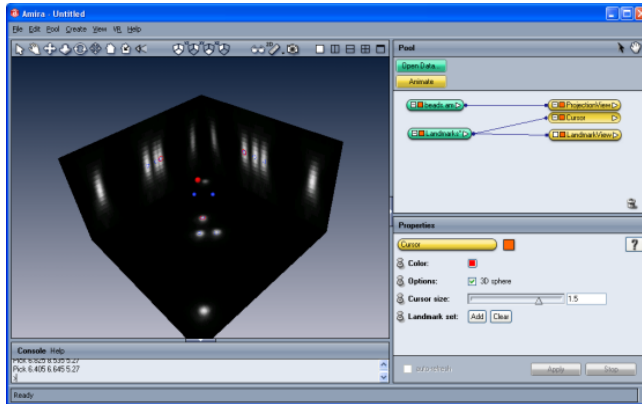
## Bead measurements

The PSF is the image of a single point source recorded under the same conditions as the actual specimen. It can be approximated by the image of a fluorescing sub-resolution microsphere, a so-called bead. Performing good bead measurements requires some practice and expertise. In order to obtain good results the following hints should be obeyed:

1. Use appropriate beads. It is important that the bead size be smaller than approximately  $1/2$  full width at half maximum (FWHM) of the PSF. Good sources for obtaining beads suitable for PSF measurements are Molecular Probes (<http://www.probes.com/>) or Polysciences (<http://www.polysciences.com/>).
2. The beads must be solid. Besides solid beads, there are also beads with the shape of a spherical shell, allowing to check the focus plane of a microscope. Such beads cannot be used as a source for PSF generation in the current version of amira.
3. Don't record clusters of multiple beads. Sometimes multiple beads may glue together, appearing as a single big bright spot. Computing a PSF from such a spot obviously leads to wrong results.
4. Note that beads are not resistant to a variety of embedding media. In particular beads will be destroyed in xylene-based embedding media such as PermOUNT (Fisher Scientific) and methyl salicylate (frequently used to clear up the tissue). As a substitute you might use immersion oil instead, which has a refractive index similar to methyl salicylate, for example.
5. Sample and beads should always be imaged as close to the coverslip as possible. When it is not possible to attach the sample to the coverslip, the beads should also be imaged in a comparable depth, embedded in the same mounting medium. Imaging the beads with better quality than the sample will yield a slightly blurred deconvolution result. However, when the PSF used for deconvolution is too wide, artifacts can arise during deconvolution.  
The objective lense should always be selected according to the mounting medium, i.e. if the sample is attached to the slide and embedded in a buffer of refractive index close to water, a severe loss of image quality can be expected when using an oil-immersion objective without a correction collar. Deconvolution of properly imaged data will always be superior to deconvolution of data suffering from aberrations.
6. Problems occur if the mounting medium remains liquid. In that case the sample distribution may not be permanent. If your specimen is to be embedded in water, you can try to immerse the beads in an agarose gel of moderate concentration instead. Attaching the small beads to the coverslip (for example by letting them dry) is often also sufficient for immobilization.

An example of an image data set containing multiple beads is provided in the file *beads.am* in the directory *Amira-4.0/data/deconv*.

- Load the data set *beads.am*.



**Figure 1.8:** Individual beads can be interactively identified using a *Cursor* module.

- Visualize the data using a *ProjectionView* module.

## Projection View and Projection View Cursor

The bead data set contains five different beads which can be clearly seen in the three orthogonal planes of the *ProjectionView* module. In order to obtain a single PSF we first want to select several “good” beads. These beads are then resampled and averaged, thus yielding the final PSF. A bead can be considered as “good” if it is clearly visible and if it is not superimposed by other beads (even when defocused),

Selecting “good” beads is an interactive process. It is most easily accomplished using the *ProjectionView*’s *Cursor* module. This module allows you to select a point in 3D space by clicking on one of the three planes of the *ProjectionView* module. The third coordinate is automatically set by looking for the voxel with the highest intensity. Points selected with the *Cursor* module can be stored in a *LandmarkSet* data object.

- Attach a *Cursor* module to the *ProjectionView* module.
- Click on any bead on one of the three planes.
- Store the current cursor position in a *LandmarkSet* object by pressing the *Add* button.
- Select and add some other beads too.

The landmarks need not to be located exactly at the center of a bead. The exact center positions can be fitted automatically later on.

You can remove incorrect bead positions from the landmark set by invoking the landmark set editor. In order to activate the editor, select the landmark set object and press Landmark Editor button. If you want to add additional bead positions to an existing landmark set object, make sure that the *master port*

of the landmark set object is connected to the *Cursor* module. Otherwise, a new landmark set object will be created.

## Resampling and averaging the beads

Now we are ready to extract and average the individual beads. This is done by means of the *BeadExtract* module.

- Connect a *BeadExtract* module to the *Landmarks* object.
- Make sure that the *Data* port of *BeadExtract* is connected to the bead data set *beads.am*. If the landmarks are still connected to the beads via the *Cursor* and *ProjectionView* modules, the connection is established automatically.

The *BeadExtract* module provides two buttons called *Adjust centers* and *Estimate size*, which should be invoked in a preprocessing step before the beads are actually extracted.

The first button (*Adjust centers*) modifies the landmark positions so that they are precisely located in the center of gravity of the individual beads.

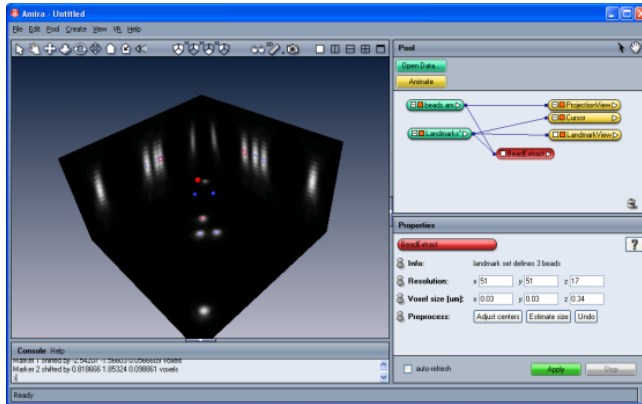
The second button (*Estimate size*) computes an estimate for the number of voxels of the PSF image to be generated. This button is only active if no PSF image is connected as a result to *BeadExtract*. If there is a result object, the resolution and voxel sizes of the result are used and the port becomes insensitive.

Any of the actions of the two preprocessing buttons can be undone using the *Undo* button. This can be necessary for example if two beads are too close so that no correct center position could be computed. In general, overlaps between neighboring beads should be avoided. Small overlaps might be tolerated because during resampling intensities are weighted according to the influence of surrounding beads.

- Perform the preprocessing steps *Adjust centers* and *Estimate size*.
- Compute a resampled and averaged PSF by pressing the *Apply* button.

The data type of the resulting PSF will be *float*, irrespective of the data type of the input image. The individual beads will be weighted on a per-voxel basis and added to the result. No normalization will be performed afterwards. You may investigate the resulting PSF image using any of the standard visualization modules. In Figure 1.10 a volume rendering of the resulting PSF is shown.

In some cases you may want to average multiple beads recorded in different input data sets. This can be easily achieved by creating a *Landmarks* object for each input data set. For the first input data set extract the beads as described above. For the other input data set also use the *BeadExtract* module. However, make sure that the PSF obtained from the first input data set is connected as a result object to *BeadExtract* before pressing the *Apply* button. In order to use an existing PSF as a result object connect the *Master* port of the PSF to *BeadExtract* (once this is done the *Resolution* and *Voxel size* ports of *BeadExtract* become insensitive, see above). If an existing result is used new beads simply will be added into the existing data set. Therefore datasets should be scaled in intensity according to

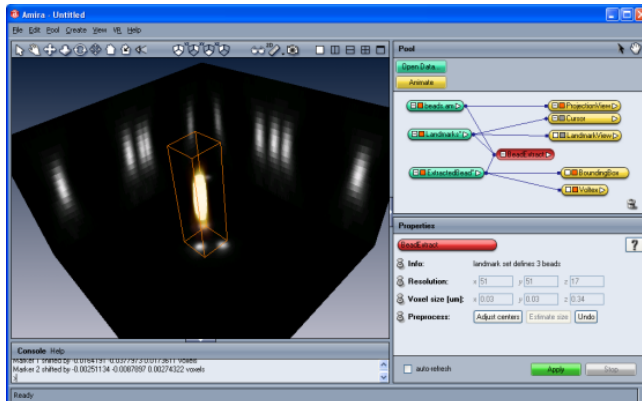


**Figure 1.9:** The *BeadExtract* module resamples and averages multiple beads.

their quality prior to bead extraction and summation to obtain a suitable weighting of the individual extracted beads in the final result.

## 1.6 Performance issues and multi-processing

Iterative maximum-likelihood deconvolution essentially is the most powerful and most robust technique for the restoration of 3D optical sections. However, it is also computationally very demanding. It can take several minutes (sometime even hours) to process large 3D data sets. This is not due an



**Figure 1.10:** The final PSF visualized using a *Voltex* module.

improper implementation but due to the algorithm itself. Both, the blind and the non-blind variant of the method, rely heavily on fast Fourier-transforms in order to efficiently compute convolutions. If you want to improve performance, try to adjust the size of your data volumes so that the number of voxels plus the border width is a power of two. Sometimes, it is worth to enlarge the border width a little bit in order to get a power of two. Although the algorithm works with data of any size, powers of two can be transformed somewhat faster.

Another issue is memory consumption. Internally, several copies of the data set need to be allocated by the deconvolution algorithm. These copies should all fit into memory at the same time (a specific variant of the algorithm suitable for working under low memory conditions will be provided in a later version). Besides the input data itself the following number of working arrays are required by the different methods:

- 3 working arrays for the non-blind algorithm with no or with fixed overrelaxation
- 5 working arrays for the non-blind algorithm with optimized overrelaxation
- 5 working arrays for the blind algorithm

The number of voxels of a working array is the product of the number of voxels of the input data set plus the border width along each spatial dimension. The primitive data type of a working array is a 4-byte floating point number. For example, if the number of voxels of the input data set plus the border width is  $256 \times 256 \times 256$  (as for the *alphalobe.am* data set in the blind deconvolution tutorial), each working array will be about 64 MB, irrespective of the primitive data type of the input data set. Therefore at least 192 MB ( $3 \times 4 \times 256 \times 256 \times 256$  bytes) are required for non-blind deconvolution with fixed overrelaxation, and 320 MB ( $5 \times 4 \times 256 \times 256 \times 256$  bytes) for blind deconvolution. Keep this in mind when configuring the computer on which to perform deconvolution! However, also note that for most platforms it usually doesn't make sense to have more than 1.5 GB of main memory. For more memory a 64-bit operating system is required.

Finally, it should be mentioned that the deconvolution algorithm can make use of a multi-processor CPU board. Although you do not get twice the performance on a dual-processor PC, a speed-up of almost 1.5 can be achieved. By default, *amira* uses as many processors as there are on the computer. If for some reason you want to use less processors you can set the environment variable `AMIRA_DECONV_NUM_THREADS` to the number of processors you actually want to use simultaneously.



## Chapter 2

# Working with Multi-Channel Images

This is a step-by-step tutorial on how to visualize multi-channel image data. To follow this tutorial you should be familiar with the basic concepts of *amira*. In particular you should be able to load files, to interact with the 3D viewer, and to connect display modules to data modules. All these issues are discussed in the *getting started* section.

We are going to load a set of multi-channel images into the workspace, attach a *MultiChannelField* group object to the data, and visualize it with several display modules. The steps are:

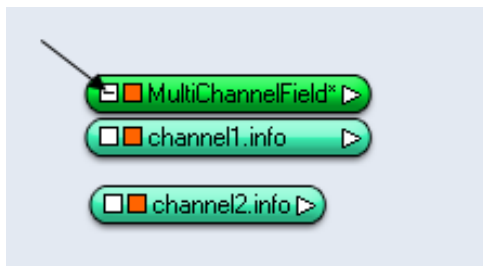
1. Load data into *amira*.
2. Create a *MultiChannelField* and attach channels to it.
3. Using *OrthoSlice* with a *MultiChannelField*.
4. Using *ProjectionView* with a *MultiChannelField*.
5. Using *Voltex* with a *MultiChannelField*.
6. Saving multi-channel images in a single AmiraMesh file.

### 2.1 Loading Multi-Channel Images into *amira*

The data we will be working with in this tutorial are confocal stacks of the prothoracic ganglion of the locust *Locusta migratoria*. They were kindly provided by Dr. Paul Stevenson, University of Leipzig, Germany. Two different channels were recorded and stored as separate files.

*amira* supports a number of proprietary multi-channel formats of several microscope manufacturers (e.g., Leica and Zeiss). In such formats all channels are stored in a single file. Therefore the first steps described in this tutorial, namely grouping the channels manually, can often be omitted.

- Load channel 1 data by selecting the file `/data/multichannel/channel1.info`



**Figure 2.1:** Data objects are connected to the *MultiChannelField* object with a right mouse click on the white square indicated by the arrow.

- Load channel 2 data by selecting the file `/data/multichannel/channel2.info`
- Create a *MultiChannelField* object by selecting *MultiChannelField* from the *Create* menu of the amira main window.

A dark green icon is displayed in the Pool. After you select the object, an info port is displayed saying "no channels connected".

- Connect channel1.info to the *MultiChannelField* by selecting 'Channel 1' from the *MultiChannelField*'s connection menu (right mouse click in the small field on the left side of the icon) and releasing it on the channel1.info icon.
- Repeat the above procedure with channel2.info

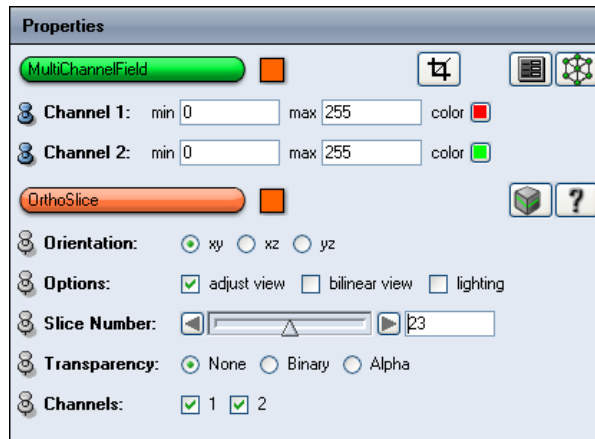
When the *MultiChannelField* is selected you will note that two entries, *channel 1* and *channel 2*, appeared in the module's control panel. Each entry has two range text fields and a color button. The range text fields work very much like those in *OrthoSlice*. Pressing the color button pops up a color dialog that lets you freely define the color of each channel. Now perhaps it is a good idea to activate the pins corresponding to Channel 1 and Channel 2 in the Properties Area. This will keep the control elements of the *MultiChannelField* module permanent in the Properties Area.

## 2.2 Using OrthoSlice with a MultiChannelField

- Connect an *OrthoSlice* module to the *MultiChannelField* by right clicking on the icon and selecting *OrthoSlice* from the context menu.

When selecting the *OrthoSlice* module, you will see that there are two additional check boxes in its control panel corresponding to the two channels. Clicking these check boxes lets you selectively switch on/off each channel. First, we adjust the intensity mapping of each channel separately.

- Switch off channel 2 by deselecting its check box.



**Figure 2.2:** When connected to a *MultiChannelField* object the *OrthoSlice* module has additional check boxes corresponding to the number of connected channels.

- Enter 23 and 200 in the min and max range fields of channel 1.

As a result, weak stainings – potentially unspecific staining – disappear and those structures that exhibit good staining become even more intense.

- Click off channel 1 and click on channel two.
- Enter the values 8 and 200 in the min and max text fields of channel 2. Move through the slices to see the results.

## 2.3 Using ProjectionView with a MultiChannelField

- Switch off the *OrthoSlice* by clicking on the viewer toggle of its icon (orange rectangle).
- Connect a *ProjectionView* module to the *MultiChannelField* by right clicking on the icon and selecting *ProjectionView* from the *Display* submenu.

As with the *OrthoSlice*, two new check boxes are shown in the module's control panel which can be used to display channels separately or simultaneously. In this way you may efficiently adjust the color and intensity of each channel before displaying them simultaneously.

## 2.4 Using Voltex with a MultiChannelField

- Switch off the *ProjectionView* by clicking on the viewer toggle of its icon (orange rectangle).

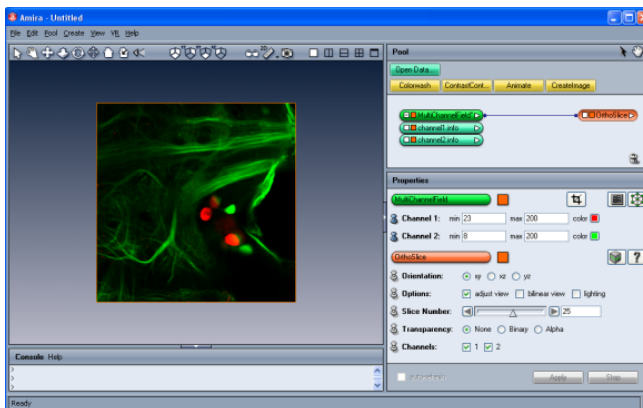


Figure 2.3: Multi channel data visualized using the *OrthoSlice* module.

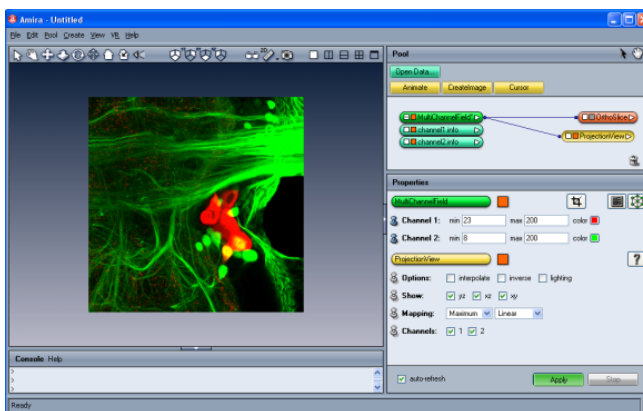


Figure 2.4: Multi channel data visualized using the *ProjectionView* module.

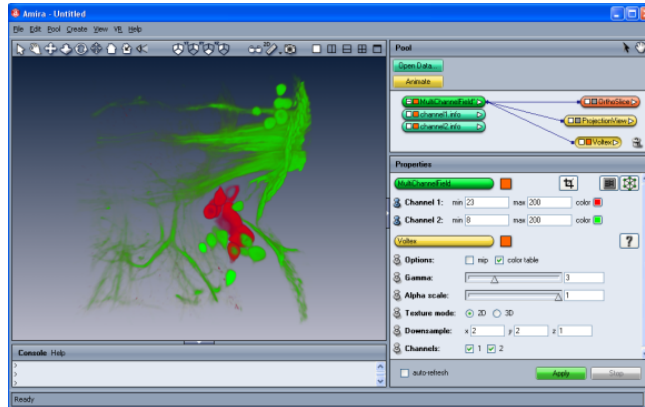


Figure 2.5: Multi channel data visualized using the *Voltex* module.

- Connect a *Voltex* module to the *MultiChannelField* by right clicking on the icon and selecting *Voltex* from the *Display* submenu.

Here also, two channel check boxes are available. Furthermore, the familiar colormap field is missing. Instead there is a slider labelled *Gamma*. Now the color of each channel is determined by that defined in the *MultiChannelField* and the *Gamma* slider controls the steepness of the alpha value (opacity) mapping used for volume rendering. Because volume rendering makes intensive use of hardware texture mapping and most consumer graphics adapters are limited in texture memory size, it is recommended to enter at least factors of 2 2 1 in the *Downsample* text fields of the *Voltex* module.

- Press the *Apply* button.

Each time you want to display another channel, you must press the *Apply* button again.

## 2.5 Saving a *MultiChannelField* in a Single AmiraMesh File

When the *MultiChannelField* icon is selected in the *Pool*, choose *Save Data As* from the *File* menu, enter a filename, and click *OK*. The data will be stored in *AmiraMesh* format so that each time you load the data the two channel stacks and the *MultiChannelField* group object will be restored.



## Chapter 3

# Skeleton Tutorial

This is a step-by-step tutorial on how to use *Large Disk Data* to analyze micro-vascular networks in human brain tissue. To follow this tutorial you should be familiar with the basic concepts of **amira**. In particular you should be able to load files, to interact with the 3D viewer, and to connect display modules to data modules. All these issues are discussed in the *getting started* section.

We are going to load 4 overlapping bricks of a large dataset. The goal is to merge these bricks into one large volume (Large Disk Data). The volume will be stored on disk only – subvolumes can be loaded into memory. Some basic operations can directly be applied to the large volume.

For the tutorial you should have access to a directory to which you're allowed to write files.

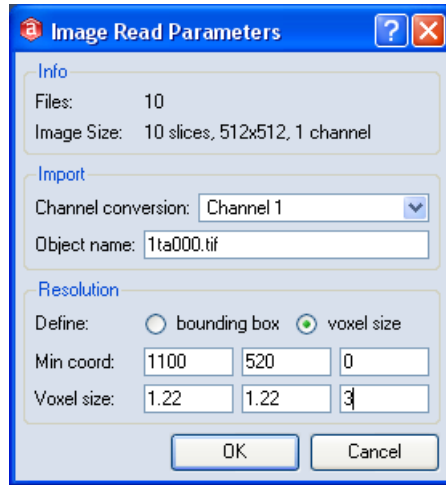
We don't provide any *TIFF* data with this tutorial. Hopefully you have a couple of blocks (*AmiraMesh* format) to test the algorithm on. Generally it is a good idea to import them into **amira** and specify an approximate bounding box near the correct position.

### 3.1 Importing your Image Data

You should have your image data as stacks of numbered 2D images. The topmost slice should have the lowest number. File formats recognized by **amira** can be found in the *File Formats* section of the user's guide. A good choice is *TIFF* because it provides lossless compression and is readable on many different systems.

You should also know the position in 3D of the lower left front corner of the brick you're going to import and the voxel size.

Choose *File/Open Data....* A *File Dialog* pops up. You can now select all of the 2D images comprising the brick. After you press *Load*, another dialog pops up:



Enter the position and the voxel size of your block. After you press *OK*, the files are loaded into one block. A new green icon will appear in the Pool. Select it and select *File/Save Data As...* to store it on disk. Name it *1ta.am*. In this way you should proceed with all of your data.

## 3.2 Arranging the Bricks

After importing your data, you should copy the files to another directory where all the processing will be done. In this way the original data is not touched and you can revert back to it if something goes wrong.

*amira* has a special data object to store links to files on disk and arrange them in 3D. It is called *Mosaic*.

- Create a *Mosaic* by selecting *Mosaic* from the *Create/Skeleton* menu of the *amira* main window.

A green icon appears. When you select it you see that it contains no bricks. The buttons below the info line are used to add data objects.

- Press the *add files* button.
- Select the files, e.g. *1ta.am*, *1tb.am*, *2ta.am*, *2tb.am* in the directory *\$AMIRA\_ROOT/data/tutorials/skeleton*. You can select multiple files at once by clicking on the first one and shift clicking on the last one.
- Press *Load*.

The selected files are added to the *Mosaic*. The Info port shows the overall number of the bricks added up to now. You can visualize the bricks with the *DisplayMosaic* module.

- Create a *DisplayMosaic* module by right clicking on the Mosaic and selecting *Skeleton/DisplayMosaic* from the context menu.
- Select the yellow DisplayMosaic icon and switch the highlighted brick by dragging the slider in the interaction area.
- Save the Mosaic.

### 3.3 Aligning Bricks

A special module allows to exactly align the bricks based on their gray values.

- Attach an *AlignBlocks* module to the Mosaic by selecting *Skeleton/AlignBlocks* in the data context menu.
- If you saved the mosaic already, its filename appears in *Mosaic name*.
- Press the *Apply* button to start the processing.

A maximum intensity projection (mip) of each block is computed. These mips are aligned and the resulting transformations are applied to the bricks on disk.

### 3.4 Filtering, Correcting Z-Drop, Resampling

It is possible to apply the same operation to all the bricks at once. To do this you must create a template for this operation. This could either be one brick with an editor (e.g. *Digital Image Filters*) attached or a compute module (e.g. *Resample*). We're going to demonstrate these two examples now. But first we should correct for the Z-Drop:

- Load one brick of the mosaic by using the *File/Open Data...* menu.
- Attach a *Deconv/CorrectZDrop* module to the brick.
- Press *Apply* to start the correction procedure and check the result.
- Attach an *ApplyTemplateToMosaic* script object by right clicking on the Mosaic and selecting *ApplyTemplate* from the *Skeleton* submenu in the context menu.
- Attach the *Template* connection of the *ApplyTemplateToMosaic* module to the *CorrectZDrop* module.
- Select *ApplyTemplateToMosaic* and press *Apply*.

Next we're going to apply a digital filter to all blocks:

- Load one brick of the mosaic by using the *File/Open Data...* menu.
- Select the data icon of the brick.
- Attach a *Digital Image Filter* by pressing the *Digital Filters* button in the Properties Area.

- Select *Gauss* from the *Filter* port, and apply it. To check the results you can attach an *OrthoSlice*.
- Attach the *Template* connection of the *ApplyTemplateToMosaic* module to the filter object (with the data attached).
- Select *ApplyTemplateToMosaic* and press *Apply*.

Another useful filter might be *Median2D* or the *Median3D*. For *Median3D*, select *Median* from the first pulldown menu of the *Filter* port, and *3D* from the second pulldown menu. The application of the latter filter takes some time but leads to good results.

The script object starts to load each brick of the mosaic, applies the filter to it, and writes it back to the same location on disk. *There are no warnings about this overwrite.*

In the next step we're going to resample every brick to an isotropic voxel size. This is only an optional step and might lead to smoother central lines in the following processing. But it increases size of the data on disk by a factor of about three. You should carefully consider whether you want to perform this step or not.

- Attach a *Resample* module to the brick loaded before and select *Mode: voxel size* and adjust *Voxel size: z* to get an isotropic result.
- Press *Apply* to start the resampling procedure and check the result.
- Attach the *Resample* module to the *Template* connection of the *ApplyTemplateToMosaic* module.
- Press *Apply* of the *ApplyTemplateToMosaic* module.

All bricks are resampled and saved to the same position. It is a good idea to sample all bricks to an isotropic voxel size. It improves the result of the distance map and the skeletonization we're going to apply.

As a standard prefiltering procedure you should:

- Apply the *ZDropCorrection*.
- Apply a 2D median filter.
- Apply a 3D Gaussian filter with a small sigma (1 or smaller).

If the results are not satisfactory, you should try to extend the prefiltering step.

## 3.5 Creating the Large Disk Data

The next step is to create a new Large Disk Data object and sample the bricks onto it. The overlapping regions can be blended with each other and a border can be added.

**Note:** The thinning algorithm expects a black border around the data. The border should be at least of size *lenOfEnds* used during thinning (see below). By default a border of 15 voxels on each side in

each dimension. Be sure to check this if you manually set *lenOfEnds*.

- Attach a *MosaicToLargeDiskData* to the Mosaic by right clicking on the Mosaic and selecting *Skeleton/MosaicToLargeDiskData* from the submenu in the context menu.
- Select the red *MosaicToLargeDiskData* icon.

You can see some options in the Properties Area. The default options are fine for the tutorial.

- A default filename derived from the mosaic is displayed in the *Filename* port. You might want to override it.
- Press the *Apply* button.

A new green icon which represents the new data object will appear in the Pool. After this the bricks will be loaded one after the other and will be sampled. This may take some time.

- Select the new green icon (titled *Image*).

In the Properties Area some information about the data stored on disk is displayed. Next,

- Delete or switch off the *DisplayMosaic* module.
- Connect a *BoundingBox* to the Mosaic icon.
- Connect a *BoundingBox* to the Image icon.

The second box is slightly bigger than the bounding box of the Mosaic. This is due to the border added by the *MosaicToLargeDiskData* module.

## 3.6 Accessing the Large Disk Data

You can't directly visualize the Large Disk Data by e.g. attaching an *OrthoSlice*. Before you can do this, you must select a subvolume and load this subvolume into the Pool. The Subvolume will be an ordinary *amira* field and you can use all the modules that you normally use. It may be a good idea to clean up the Pool now, but it's not required. The Mosaic is no longer needed.

- Connect an *Access* to the Image object by right clicking on it and selecting *Access* from the popup menu.
- Select the red *Access* icon.

In the viewer you can see a dragger box in one corner of the bounding box of the Large Disk Data. You can click and drag the corners or the faces of the box to specify the subvolume you want to load. In the Properties Area the corresponding dimensions are displayed.

- Drag the box somewhere inside the volume (For this you need to switch the viewer into interaction mode).
- Press the *Apply* button.
- Attach an OrthoSlice to the new green icon (Image.view).
- Select the Access object, then toggle on the *auto-refresh* check box.
- Drag the box in the viewer.

By setting *auto-refresh* on, every time you drag the box an automatic reload is started and all modules downstream of the view are recomputed. This is an easy way to scan through the large volume. Try different display modules on the Image.view, e.g., an isosurface.

## 3.7 Computing directly on the Large Disk Data

Some computation modules are able to handle the Large Disk Data directly. These include thresholding, computation of a distance map, thinning, extracting a line set from a voxel skeleton, and computation of the thickness of the lines (evaluating the distance map at the points of the lineset). All these steps are presented in this subsection.

The first step is to apply a simple thresholding.

- Attach a *Threshold* to the *Image* icon by right clicking on it and selecting *Threshold* from the *Skeleton* submenu in the popup menu.
- Select an appropriate threshold in the Properties Area.
- Select a filename you want to store the result to. In the tutorial we will use the default name *Image.labels*.
- Press the *Apply* button.

A new green icon that contains the labels will appear. Connect an *Access* module to it as described above and have a look at the results.

You might want to correct the result of the segmentation procedure manually. This might be useful to fill big vessels or remove uninteresting parts. *amira* has a *segmentation editor* to perform this task. Due to the size of the data, you will have to work on subblocks of the whole data set.

In the next step we'll calculate a distance map of the object.

- Attach a *ChamferMap* to the *Image.labels* icon by right clicking on it and selecting *ChamferMap* from the *Skeleton* submenu in the popup menu.
- Specify a filename (the default is OK for the tutorial).
- Press *Apply*.

A new green icon named Image.dm will appear. Connect an *Access* module and have a look at the

distance map.

The thinning procedure needs the labels and the distance map as input.

**Note:** The thinning algorithm automatically detects endpoints of vessels. A parameter is used to distinguish them from "noise" on the surface of the vessels to avoid spurious branches. You might want to change this parameter manually in the console. Use `Thinner setVar lenOfEnds 10` to set the length of the ends to 10 voxels before they are detected as unconnected ends. This is a rather large value leading to only a few branches. The drawback is that you also might miss real endpoints. It will be really hard to detect such errors during the network check. But in general we think it is a good idea to avoid spurious branches directly during thinning.

- Attach a *Thinner* to the *Image.labels* icon by right clicking on it and selecting *Thinner* from the *Skeleton* submenu in the popup menu.
- Connect the port for the distance map to the *Image.dm* icon. You can achieve this by right clicking on the white square on the left side of the *ExtThinner* icon and selecting *Distmap*. A blue line is attached to the mouse pointer; after you click on the *Image.dm* icon, the two modules are connected.
- Specify a filename (the default is fine for the tutorial).
- Press *Apply*.

A new green icon named *Image.thinned* will appear and the thinning process is started. It may take some time before it finishes. We will directly go on and convert the result into a lineset before visualizing it.

- Attach a *TraceLines* to the *Image.thinned* icon by right clicking on it and selecting *TraceLines* from the *Skeleton* submenu in the popup menu.
- Unselect the *cluster* toggle in the Properties Area.
- Press *Apply*.

The new icon that is visible now in the Pool is a lineset, which you are probably already familiar with. You can visualize it by connecting a *LineSetView*.

- Create an *LineSetView* module by right clicking on the *Image.lineset* and selecting *LineSetView* from the context menu.

The lines are rather jaggy because they connect centers of voxels. To get smoother lines you can use a TCL command in the console.

- Type `Image.lineset smooth` into the amira console window. You can repeat this if you would prefer even smoother lines.

You can use the *CheckNetwork* module from the *Skeleton* submenu in the context menu to remove short ends.

In the last step of this subsection we will compute a thickness for every point on the lines. For the thickness we use the values of the distance map.

- Attach an *EvalOnLines* to the *Image.lineset* icon by right clicking on it and selecting *EvalOnLines* from the *Compute* submenu in the popup menu.
- Connect the port for the distance map to the *Image.dm* icon. You can achieve this by right clicking on the white square on the left side of the *EvalOnLines* icon and selecting *Field*. A blue line is attached to the mouse pointer; after you click on the *Image.dm* icon, the two modules are connected.
- Press *Apply*.

The module doesn't create a new data icon. It is more like an editor and changes the connected lineset. It adds a data value for every vertex in the lineset and calculates the value of the field at the point of the vertex. You can visualize the data with the *LineSetView*.

- Select the *LineSetView* by clicking on it.
- In the Properties Area there is a drop down menu called *ColorMode*. Click on it and select *Data 0*.
- Right click on the rectangular area in the row *Colormap*. A popup menu appears. Select *physics.icol*.
- Change the range of the colormap by clicking into text field right of the colormap and type in 15.

You see that the lines are now colored. The color is an indicator for the local radius of the original object.

## 3.8 Region of Interest

During visualization of large datasets there is often the need to restrict the displayed geometry to a subvolume of the total dataset. It would be nice if different modules shared the same volume and the volume could be changed simultaneously for all of them. In *amira* there is a special module that provides this possibility; it is called *SelectRoi*. You can attach it to every spatial data object. Some display modules have a connection called *ROI* that can be attached to the *SelectRoi* module to restrict the view.

- Remove all objects except for the *Image.lineset* and the *LineSetView*.
- Create a *SelectRoi* module by right clicking on the *Image.lineset* and selecting *SelectRoi* from the *Display* submenu of the context menu.
- Connect the Connection named *ROI* of the *LineSetView* to the *SelectRoi* module. To do this, right click on the white square on the left side of the *LineSetView* icon and select *ROI*. A blue

line is attached to the mouse pointer; after you click on the SelectRoi icon, the two modules are connected.

- Switch the viewer to interaction mode and click and drag one of the green squares. This will adjust the Region of Interest and the LineSetView will adopt the new restriction immediately.
- By clicking and dragging on the (invisible) faces of the cuboid you can move it to another position.

When working with a small subset of the lineset, it is possible to do more involved visualizations that require more graphics power. For example, the lines can be displayed as tubes that reflect the local thickness.

- Choose a rather small part of the lineset.
- Select the *LineSetView*.
- Click on the *Shape* drop down menu and select *Circle*.
- Click on the *ScaleMode* drop down menu and select *Data 0*.
- Move the *ScaleFactor* slider to 2.

In the viewer the lines are now displayed as tubes. The thickness is scaled with the data associated with the lines.

**Note:** The data value associated with the lines is the local radius. The LineSetView scales by the local diameter. To scale to the physical size you therefore must use a ScaleFactor of 2.

In the next step we're going to load a part of the image data that is also determined by the SelectRoi module. You can then easily load the same subvolume from different lda files if you connect all the Access modules to one common SelectRoi module.

- Load the file *Image* that you saved before.
- Attach an Access module to it (see above if you don't know how).
- Connect the Connection named ROI of the Access to the SelectRoi module. This is done the same way as with the LineSetView.
- Select the Access module, then press the *Apply* button.
- Attach a ProjectionView display module to the newly displayed green *Image.view* icon.
- You can do the same for the *Image.labels* file.

All the LargeDataAccess modules you created are now restricted to the same volume and can easily be moved by one click-and-drag operation.

## 3.9 Check Network

In this subsection we'd like to present a module that can be used to jump to all endpoints of a lineset and create some nice views for checking if the endpoints are fine or if they should be edited.

- Create a *CheckNetwork* module by right clicking on the *Image.lineset* and selecting *CheckNetwork* from the *Skeleton* submenu of the popup menu.
- Connect the *SelectRoi* connection of the *CheckNetwork* to the *SelectRoi* module (right click on the white square at the left of *CheckNetwork*, select *SelectRoi*, click on the yellow *SelectRoi* icon).
- Select the *Access* module, and toggle on the *auto-refresh* check box.
- Adjust the size of the lines by selecting the *LineSetView* and changing the *ScaleFactor* slider to approximately 0.15.
- Select the *CheckNetwork* module.
- Press the *Next Endpoint* button.
- By repeating the last step you can jump through all endpoints.

## 3.10 Coloring a Lineset According to its Depth Value

It can be useful to color the lines in different ways. In the next example we're going to color the lineset by the local z value. This is done in two steps:

- Create a *Scalarfield* which provides the depth (z value).
- Evaluate this value on the lines and use a *LineSetView*.

To create the *Scalarfield*:

- Select *Create/Data/Scalarfield*.
- Select the newly created icon.
- Type *z* into the *Expr* field.

The next step is to evaluate this scalarfield on the lineset. You can do this by selecting the lineset and typing into the console.

**Hint:** Press the <TAB> key to get the name of the selected module.

- Type `lines computeData scalarfield 2` to evaluate the data (Fill in your specific names for `lines` and `scalarfield`). The 2 indicates to store the data values as data 2 in the lineset.
- Attach a *LineSetView* and use *data2* for color coding.

## **Part II**

# **Reference Guide - Alphabetic Index of Modules**



# Chapter 4

## ResolveRT

### 4.1 BeadExtract

This module can be used to resample and average the image of one or multiple beads, i.e., fluorescing sub-resolution microspheres, thereby obtaining an approximation of a point spread function (PSF) required for non-blind deconvolution.

The module must be connected to the image data set containing the measured beads as well as to a landmark set indicating the center positions of all beads to be resampled and averaged. The whole process of obtaining a point spread function from a bead measurement is described in a separate *tutorial on bead extraction*. Please refer to this tutorial for more information on how to use the module.

Press the *Apply* button to actually extract the beads with the center of the resulting dataset corresponding to the current positions of the landmarks. The image volume around every landmark is extracted and resampled using a Lanczos filter (compare the *Resample* module). The resampled bead images are then added to the result object. The final PSF is not normalized.

### Connections

#### **Data** [required]

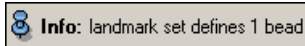
Connection to a uniform image data set containing measured beads.

#### **Landmarks** [required]

Connection to a landmark set indicating the center positions of the beads to be resampled and averaged.

## Ports

### Info



Displays the number of beads to be resampled and averaged.

### Resolution



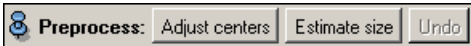
Specifies the number of voxels of the final PSF image to be generated. If a PSF image is connected as a result object to this module, the port becomes insensitive and the number of voxels of the result object are used.

### Voxel size [um]



Specifies the voxel size of the final PSF image to be generated in micrometers. If a PSF image is connected as a result to this module, the port becomes insensitive and the voxel size of the result object is used.

### Preprocess



*Adjust centers:* If this button is pressed, the landmark positions of the data set connected to port *Landmarks* are shifted to the center of gravity of the corresponding bead. It is required that the initial landmark positions be inside the bead images and that neighboring beads do not overlap significantly. Otherwise incorrect center positions may be computed.

*Estimate size:* If this button is pressed, the desired number of voxels of the final PSF image specified in port *Resolution* are computed automatically. Before this is done the beads' center positions already should have been adjusted. The estimate is computed by determining the extent of the biggest spot around any landmark. Again, it is required that neighboring beads do not overlap significantly. The *Estimate size* button becomes insensitive if a result object is connected to the module. In this case always the size of the existing PSF image will be used.

*Undo:* Undoes the effect of any of the previous two buttons. For example, if wrong center positions have been computed after pressing *Adjust centers*, the original landmark positions can be restored using the *Undo* button.

## 4.2 Convolution

This module convolves two uniform 3D data objects with each other by Fourier transforming the two inputs, multiplying them, and then transforming them back. The module is part of the *amira deconvolution modules*. It can be used for example to verify the results of image deconvolution.

The bounding box and voxel sizes of the input data set and the convolution kernel are ignored by this module. Use the *Resample* module to make sure that the resolution of both inputs is identical.

Press the *Apply* button to start the computation.

## Connections

**Data** [required]

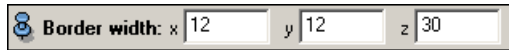
The data set to be convolved.

**Kernel** [required]

The convolution kernel.

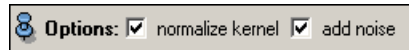
## Ports

### Border width

A port control for 'Border width' with three input fields for x, y, and z dimensions. The x and y fields are set to 12, and the z field is set to 30. Each field has a small blue icon to its left.

Defines the size of the border region. A border region is necessary if the input data set doesn't fade out to black at the boundaries.

### Options

A port control for 'Options' with two checked checkboxes: 'normalize kernel' and 'add noise'. Each checkbox has a small blue icon to its left.

If *normalize kernel* is selected, the integral of the convolution kernel (connected to port *Kernel*) will be normalized to one. If this is not the case, the intensities of the convolved data set will be scaled by the actual integral.

If *apply noise* is selected, the values of the convolved data set will be multiplied by random numbers uniformly distributed around 1 (white noise).

### Noise level

A port control for 'Noise level' with a slider and a numeric input field. The slider is set to 0, and the input field also shows 0. Each field has a small blue icon to its left.

This port will only be shown if the *apply noise* option of the *Options* port has been selected. It specifies the amount of noise applied to the output, i.e., the range of the random numbers around 1, by which the result is multiplied.

## 4.3 CorrectZDrop

This module lets you fix artifacts in 3D microscopic images caused by light absorption in other slices. If such artifacts are present, the average intensity in lower slices seems to be decreased. This so-called *z-drop* or *intensity attenuation* can be corrected automatically by fitting an exponential curve to the average intensities in each of the slices, or manually by providing a user-defined formula.

Press the *Apply* button to start the computation.

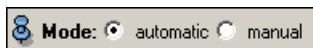
## Connections

### Data [required]

The image data exhibiting a z-drop artifact. Scalar fields with uniform or stacked coordinates as well as *multi-channel fields* are supported.

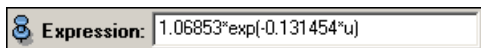
## Ports

### Mode



Lets you select between *automatic* mode and *manual* mode.

### Expression



This port is only available if *manual* mode has been selected. It provides a text field where you can enter a formula specifying a factor used to multiply the intensity values in each slice. Within the formula the variable  $u$  specifies the slices.  $u$  will take the value 0 for the first slice and 1 for the last slice. For the other slices it takes intermediate values depending on the actual slice location (this makes support of stacked coordinates easy). In automatic mode the following formula  $a \cdot \exp(b \cdot u)$  will be used, where  $a$  and  $b$  are fitted automatically. If you first perform an automatic z-drop correction and then switch to manual mode, the fitted exponential will be displayed in the port's text field.

## 4.4 DataPreprocess

This module can be used to apply both a background and a flatfield correction to a raw 3D image stack.

For the background correction, a single background image must be provided at the *background* port of the module. The background image should be a nearly black image recorded with the camera's shutter closed. This image is subtracted from all slices of the 3D input data set, thus compensating for any dark current of the camera's CCD detector.

For the flatfield correction, a single flatfield image must be provided at the *flatfield* port of the module. The flatfield image should be an unfocused almost white image taken from a drop of homogeneously fluorescing dye. The intensities of the 3D input image are then scaled according to the normalized intensities of the flatfield images. The input image gets brighter at pixels where the flatfield is dark and vice versa. In this way non-uniform sensitivity of the camera's CCD detector is compensated for. If both a flatfield and a background image are present, the background is subtracted from the flatfield too.

Press the *Apply* button to start the computation and produce a corrected output data set.

## Connections

### Data [required]

The raw 3D image stack to be corrected. Any regular scalar field with uniform coordinates is supported.

### Background [optional]

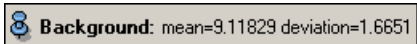
An optional 2D background image with the same number of voxels in the x and y directions as the 3D input image. If an input is present at this port, a background correction is performed (see above).

### Flatfield [optional]

An optional 2D flatfield image with the same number of voxels in the x and y directions as the input image. If an input is present at this port, a flatfield correction is performed (see above).

## Ports

### Background



Displays the mean value and the standard deviation of the background image, if such an image is present. Only the first slice is considered.

### Flatfield



Displays the mean value and the standard deviation of the flatfield image, if such an image is present. Only the first slice is considered.

## 4.5 Deconvolution

This module is the front-end for deconvolving 3D microscopic images. Two different iterative maximum-likelihood image restoration algorithms are provided, a non-blind one and a blind one. For a general description of the deconvolution process, please refer to the provided *tutorials*.

The resulting deconvolved data set will be stored in the Pool. If no input PSF is specified or if the blind deconvolution algorithm has been selected, the estimated PSF will be stored in the Pool also.

Press the *Apply* button to start the deconvolution process. Since deconvolution is a time consuming operation, it optionally can be performed as a batch job (see the *Action* port below).

## Connections

### Data [required]

The data set to be deconvolved.

## Kernel [optional]

The point spread function (PSF) to used for deconvolution. If no PSF is specified, an estimated PSF is calculated automatically based on the numerical aperture of the microscope, the wavelength of the emitted light, and the refractive index. If a blind deconvolution is to be performed, an input PSF (if connected) will be used as an initial estimate.

## Ports

### Border width

A control panel for border width. It features a blue icon with a white 'S' on the left. To its right is the text 'Border width:'. Following this are three input fields: 'x' with the value '8', 'y' with the value '8', and 'z' with the value '67'.

Defines the size of the border region. A border region is necessary if the input data set doesn't fade out to black at the boundaries. For performance reasons it might advisable to choose values such that the sum of the size of the input data set and the border width results in a power of two. For example, if the data set consists of 118 slices, a border width of 10 slices in z direction would be a good choice.

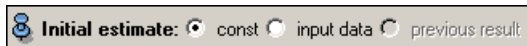
In the case of widefield data it is sometimes advisable to have the border width in z direction exactly as large as the data set itself. If this is the case, the border region will be initialized by mirroring the values from the actual data volume. Otherwise, the values of the first slice and of the last slice will be interpolated linearly.

### Iterations

A control panel for iterations. It features a blue icon with a white 'S' on the left. To its right is the text 'Iterations:'. This is followed by a slider control with a triangle in the center, and a numeric input field on the right containing the value '60'.

Specifies the number of iterations of the deconvolution procedure.

### Initial estimate

A control panel for the initial estimate. It features a blue icon with a white 'S' on the left. To its right is the text 'Initial estimate:'. This is followed by three radio button options: 'const' (which is selected), 'input data', and 'previous result'.

Specifies the initial estimate of the deconvolution algorithm. If *const* is chosen, a constant image is used initially. Often, this yields smoother results than the second option, namely *input data*. The third option (*previous result*) is only available if the input data set has already been deconvolved previously. Use this option if you want to apply some additional deconvolution iterations.

### Overrelaxation

A control panel for overrelaxation. It features a blue icon with a white 'S' on the left. To its right is the text 'Overrelaxation:'. This is followed by three radio button options: 'none', 'fixed' (which is selected), and 'optimized'.

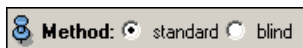
Overrelaxation is a technique to speed up the convergence of the iterative deconvolution process. In most cases *fixed* overrelaxation is a good choice. For non-blind deconvolution also an *optimized* overrelaxation technique is available. This method further accelerates convergence but is more memory and time consuming.

### Regularization



This port specifies if you want *intensity-based* regularization or not.

## Method



This port specifies whether standard (non-blind) or blind deconvolution should be used.

## PSF Parameters



Parameters for calculating the point spread function. This port will only be shown if standard (non-blind) deconvolution has been selected and no input PSF was specified, or if blind deconvolution has been selected, in which case these parameters act as constraints.

*NA* denotes the numerical aperture of the microscope. *lambda* denotes the wavelength of the emitted light in micrometers with the voxel sizes also being interpreted in micrometers. Finally, *n* denotes the refractive index of the specimen.

## Microscopic Mode



Selects whether the input image has been recorded using a widefield microscope or using a confocal microscope. This is important for the PSF generation as well as for the selection of appropriate constraints during blind deconvolution.

## Action



Since deconvolution is a time consuming operation, it optionally can be performed as a batch job. A batch job can be submitted using the *Batch job* button. If this button is pressed, first a dialog is popped up allowing you to specify the filename of the final deconvolved data set as well as the number of optional check point files (compare Figure 1.6). A check point stores an intermediate result obtained after a certain number of iterations have been performed. The actual deconvolution job is started via the job dialog accessed by *File / Jobs* menu item. The job dialog is popped up automatically after the job has been finally submitted, but this may take a few seconds if there are currently no jobs running.

## 4.6 DistanceMap

This module computes a 3D distance field of a 3D object. Each voxel will be assigned a value depending on the distance to the nearest object boundary. The boundary voxels of the object are assigned a value of zero whereas the assigned value increases as the distance increases.

To use this module it must be connected to a uniform label field where each voxel with a nonzero value is assumed to belong to the object.

Press the *Apply* button to start the computation.

## Connections

**Data** [required]

Labelfield from which the distance map is computed.

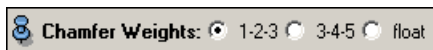
## Ports

### Type

A control panel for the 'Type' port. It features a blue icon with a gear and a plus sign on the left. To its right is the label 'Type:' followed by three radio button options: 'Euclid', 'Chamfer', and 'SingleSeeded'. The 'Chamfer' option is currently selected.

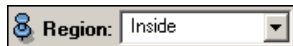
You may either choose a true Euclidean distance metric or an approximation based on a 3x3x3 chamfer metric. The latter is much faster to compute and accurate enough for most applications. Single Seeded computes a distance map which expresses the distance from a single seed point rather than from the boundary.

### Chamfer Weights

A control panel for the 'Chamfer Weights' port. It features a blue icon with a gear and a plus sign on the left. To its right is the label 'Chamfer Weights:' followed by three radio button options: '1-2-3', '3-4-5', and 'float'. The '1-2-3' option is currently selected.

This port is only available in chamfer mode. Different chamfer metrics are available. The 1-2-3 metric is equivalent to only considering a 6-neighborhood when propagating the distance value, whereas the 3-4-5 considers a 26-neighborhood and is a better approximation of the Euclidean distance metric. Float also corresponds to a 26-neighborhood but the resulting field will have float data type instead of short int.

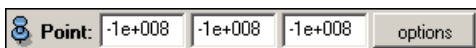
### Region

A control panel for the 'Region' port. It features a blue icon with a gear and a plus sign on the left. To its right is the label 'Region:' followed by a dropdown menu currently showing 'Inside'.

This port is not available in Single Seeded mode. Choose in which region the distance field will be computed:

- **Inside:** Inside the object (outside will be set to zero).
- **Outside:** Outside the object (inside will be set to zero).
- **Both (unsigned):** Inside and outside the object. A positive distance is computed whether the position is inside or outside the object.
- **Both (signed):** The distance value will be negative at a position inside the object and positive outside the object.

### Point

A control panel for the 'Point' port. It features a blue icon with a gear and a plus sign on the left. To its right is the label 'Point:' followed by three input fields, each containing '-1e+008', and an 'options' button.

This port is only available in Single Seeded mode. Specifies the seed point for the distance map in world coordinates. You can use a dragger to adjust it.

## 4.7 FourierTransform

This module computes a discrete forward or backward Fourier transform from a scalar input data set with uniform coordinates. Alternatively, the power spectrum, i.e., the squared magnitude of the Fourier transform can be computed.

The origin of the input data set will be ignored by this module. Also, instead of being expressed in wave numbers, the bounding box of a Fourier transformed data set will be the same as the bounding box of the input.

Press the *Apply* button to start the computation.

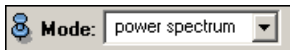
### Connections

**Data** [required]

The input data to be Fourier transformed.

### Ports

**Mode**



Option menu specifying the action to be performed.

If a real scalar field is connected to the module, three options are available, namely *forward*, *forward complex*, and *power spectrum*. If *forward* is selected, the output is stored in a special so-called half-complex format. Such an output can be back-transformed, but not many other operations can be performed on it. If *forward complex* is selected, the output will be a complex-valued scalar field with the same number of voxels as the input object. Finally, if *power spectrum* is selected, the output is a real-valued scalar field with the same number of voxels as the input object containing the squared magnitude of the Fourier transform.

If a complex scalar field in half-complex format is connected, the only option is *backward*, indicating a backward Fourier-transform.

If an ordinary complex scalar field is connected to the module, the three options *forward*, *backward*, and *power spectrum* are available. The first two options specify a forward and backward Fourier transform, respectively. The output is a complex scalar field with the same resolution as the input. If *power spectrum* is selected, the output is a real-valued scalar field with the same number of voxels as the input object containing the squared magnitude of the Fourier transform.

## 4.8 PSFGen

This module can be used to compute a point spread function (PSF) for deconvolution of widefield and confocal microscopic image data. PSF computation is based on electromagnetic vector theory. It can be created by selecting it from the *Create / Others* menu of the main window.

Press the *Apply* button to compute the PSF.

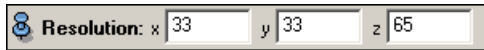
### Connections

#### Data [optional]

A uniform scalar field (usually the image data to be deconvolved) can be connected to this port. The values of the *Resolution* and *Voxel size* ports will be set automatically to the values of the input field then.

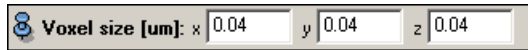
### Ports

#### Resolution

A control panel for the Resolution port. It features a blue circular icon with a white 'S' on the left. To its right is the label 'Resolution:'. Further right are three input fields: 'x' with the value '33', 'y' with the value '33', and 'z' with the value '65'.

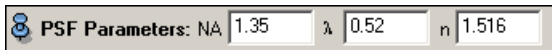
The number of voxels of the PSF image to be generated.

#### Voxel size [um]

A control panel for the Voxel size port. It features a blue circular icon with a white 'S' on the left. To its right is the label 'Voxel size [um]:'. Further right are three input fields: 'x' with the value '0.04', 'y' with the value '0.04', and 'z' with the value '0.04'.

The voxel size in microns of the PSF image to be generated.

#### PSF Parameters

A control panel for the PSF Parameters port. It features a blue circular icon with a white 'S' on the left. To its right is the label 'PSF Parameters:'. Further right are three input fields: 'NA' with the value '1.35', ' $\lambda$ ' with the value '0.52', and 'n' with the value '1.516'.

Parameters for calculating the point spread function. *NA* denotes the numerical aperture of the microscope. *lambda* denotes the wavelength (as measured in a vacuum) of the emitted light in micrometers. For confocal data the excitation and emission wavelength are assumed to be identical. In this case it might prove useful to compensate by supplying a value between excitation and emission wavelength as parameter. Finally, *n* denotes the refractive index of the specimen.

#### Microscopic Mode

A control panel for the Microscopic Mode port. It features a blue circular icon with a white 'S' on the left. To its right is the label 'Microscopic Mode:'. Further right are two radio buttons: 'widefield' (which is selected) and 'confocal'.

Specifies whether the PSF of a widefield microscope or of a confocal microscope should be computed.

# Chapter 5

## Skeleton Pack

### 5.1 AlignBlocks

The *AlignBlocks* module is for aligning the blocks of a mosaic. During the acquisition of the image blocks of a mosaic, it may happen that the blocks are not perfectly aligned. This module computes the best adjustment between overlapping blocks, i.e., finds the translations between each pair of blocks that minimizes the difference in the common part of the two images.

Press the *Apply* button to start the computation.

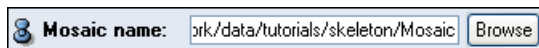
#### Connections

**Data** [required]

*Mosaic*

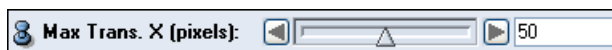
#### Ports

**Mosaic name**



Stores the filename of the final mosaic and of the temporary files: during the planar alignment, the module creates 2D projections of the image blocks. These projections are stored with the extension "nb.mip.am" where *nb* represents the number of the block in the mosaic.

**Max Trans. X (pixels)**



Maximum value of translation to apply in the X direction (in pixels).

### Image units value

 **Image units value:** 30.000000 (image unit)

Gives the corresponding value in image units (for example, if the voxel size is given in micrometers, gives the maximum translation in micrometers).

### Max Trans. Y (pixels)

 **Max Trans. Y (pixels):**  50

Maximum value of translation to apply in the Y direction (in pixels).

### Image units value

 **Image units value:** 30.000000 (image unit)

Gives the corresponding value in image units (for example, if the voxel size is given in micrometers, gives the maximum translation in micrometers).

### Max Trans. Z (pixels)

 **Max Trans. Z (pixels):**  10

Maximum value of translation to apply in the Z direction (in pixels).

### Image units value

 **Image units value:** 30.000000 (image unit)

Gives the corresponding value in image units (for example, if the voxel size is given in micrometers, gives the maximum translation in micrometers).

### Alignment

 **Alignment:** ☒ planar ☒ vertical

Performs a planar and/or a vertical alignment. If *planar* is checked, makes a 2D projection of each image (on XY plane), and finds the best planar translation between each image. If *vertical* is checked, then finds the vertical component of the translation.

### Similarity measure

 **Similarity measure:** ☐ SSD ☒ Correlation

This port allows you to select either the *SSD* (sum of squared differences) or the *correlation* method for computing how well the blocks are aligned.

## 5.2 ApplyMask

This module can be used to segment a LargeDiskData file block by block. It allows you to apply a mask to the LargeDiskData file in order to remove or add parts.

Attach it to the LargeDiskData file containing the labels and attach a Labelfield containing the mask to the second input connection.

After selecting which materials in the Labelfield describe regions to be added (set to material 1) in the LargeDiskData and which regions should be deleted (set to background), press the *Apply* button.

## Connections

### Data [required]

The connection to the LargeDiskData object.

### Mask [required]

The connection to a Labelfield containing the mask.

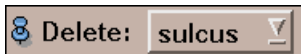
## Ports

### Add



Select the material describing the region which will be set to 1 in the LargeDiskData object.

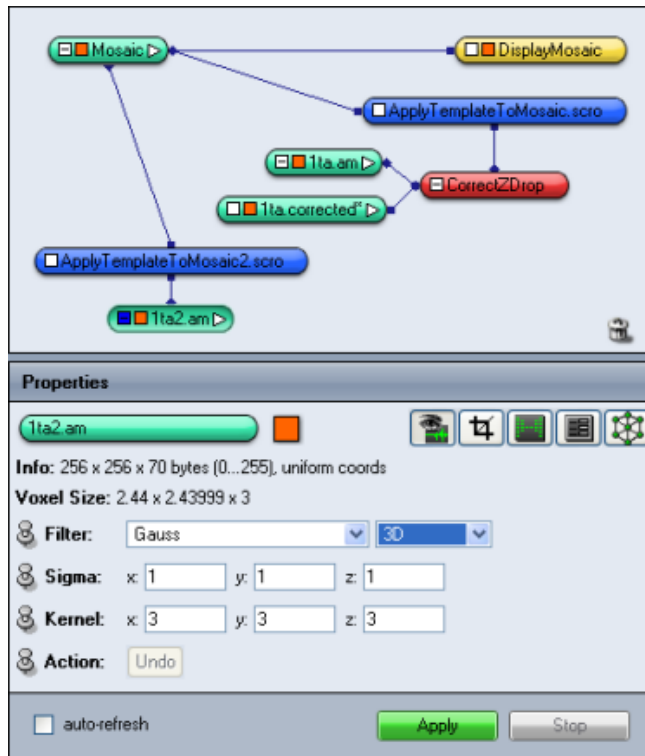
### Delete



Select the material describing the region which will be set to 0 in the LargeDiskData object.

## 5.3 ApplyTemplateToMosaic

This script object is useful if you want to apply the same compute module or *digital image filter* to each brick of a mosaic. You will need to build a network as shown in the figure:



If you plan to apply a compute module, e.g., *CorrectZDrop*, load one brick and apply the compute module to this brick. Connect the *Template* input port of *ApplyTemplateToMosaic* to the compute module. To start the processing, press *Apply*.

If you want to apply a *digital image filter* to every brick, load one block, start the editor on it, and select the parameters. Connect the *Template* input port to this data object and press *Apply* to run the filter on all bricks.

## Connections

### Data [required]

The *Mosaic* to be processed.

### Template [required]

The module providing the template which will be applied to all bricks.

## 5.4 ChamferMap

The *ChamferMap* module performs a distance-map calculation on a 3D segmented *LargeDiskData* image. It creates a new *LargeDiskData* object where the value of each point of the foreground represents its shortest distance to the background or each background point represents the shortest distance to the foreground, depending on the option selected in the *Map location* port.

The chamfer map is first computed with integer values, which causes the real values of distances to be scaled by a coefficient (see *ChamferMapScaleFactor*). If you want the units of your chamfer map to be the same as the units of your voxels, check the *Float exact map* box.

For isotropic images, you can choose the dimension and the size of the chamfer mask (its coefficients are pre-calculated). For anisotropic images, however, it first computes the chamfer mask coefficients that are most appropriate for the voxel size of your image. For this reason, using a mask wider than 3x3x3 for anisotropic images is strongly discouraged.

If the original image is not segmented, it will consider positive points as foreground and negative points as background.

Press the *Apply* button to start the computation.

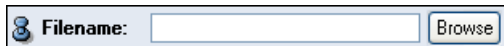
### Connections

**Data** [required]

*LargeDiskData* (should be segmented).

### Ports

**Filename**



The result chamfer map will be saved as a *LargeDiskData*, but with additional parameters in the parameter database:

**ChamferMapScaleFactor:** factor to scale your chamfer map with the same units as your voxel units. For example, if the voxel size of your image is expressed in micrometers, divide your distance map by the *ChamferMapScaleFactor* and you will obtain a chamfer map in micrometers.

**ChamferMapMaxRelError:** gives the max relative error relative to the exact Euclidean distance.

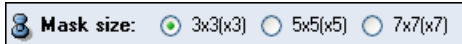
**ChamferMapIsExact:** is set to 1 if *ChamferMapScaleFactor* = 1; is set to 0 otherwise.

### Mask dimension



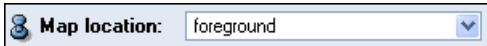
Choose 2D if you want the chamfer map to be computed slice by slice without considering the previous and following slices. This option is not available for anisotropic images.

### Mask size



The mask size for isotropic images. The wider the chamfer mask, the more precise the chamfer map, but the longer it takes to compute. This option is not available for anisotropic images.

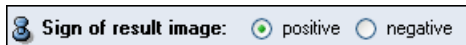
### Map location



This option menu lets you select different map locations:

- **foreground:** Compute the minimum distance of each point of the foreground relative to the background
- **background:** Compute the minimum distance of each point of the background relative to the foreground
- **both foreground and background:** Compute the minimum distance of each point of the foreground relative to the background, with positive values; and the minimum distance of each point of the background relative to the foreground, with negative values.

### Sign of result image



Determines the sign of the points of the foreground if you have chosen *foreground* in the *Map location* port, or the sign of the points of the background if you have chosen *background*.

### Float exact map



Computes a distance map in the same units as the voxel size units.

## 5.5 CheckNetwork

The module detects open ends and branching points with more than 3 branches in an attached *Lineset*. It sorts these points and moves a *SelectRoi* object to be centered at one point after another. The focus is attracted to the selected point by a red semi transparent sphere.

## Connections

**Data** [required]

LineSet.

**SelectRoi** [optional]

Roi (Region of Interest) to be moved.

## Ports

### Info

 **Info:** jumped to endpoint 4 of 125

Shows information about the last step.

### Options

 **Options:** ☐ long only ☒ open Surfaces ☒ ignore Border

Select *long only* to ignore short lines.

If *open Surfaces* is selected, the module will cut away half of each Isosurface attached in some way to the Roi. This may be useful to look inside the structure to see the lines you're checking.

*ignore Border* forces the module to skip points near the boundary of the bounding box.

### Limits

 **Limits:** border

Fraction of the length of one side of the bounding box to be ignored on each side. 0 means take all, 0.5 ignores everything.

### Action

 **Action:**

*next unvisited EP* jumps to next endpoint.

*next unvisited BP* jumps to next branching point.

### BufferAction

 **BufferAction:**

Each visited point is stored in a buffer and can be visited again.

Use these buttons to navigate through the buffer.

### OtherAction

 **OtherAction:**

Every point already visited is marked in the lineset. *clear visited* removes these markers.

*rm unconn lines* keeps only the largest connected component in the lineset.

## 5.6 DisplayMosaic

Attach this module to a *Mosaic* to display its bricks.

Press the *Apply* button to load the highlighted brick into the Pool.

### Connections

**Data** [required]

The *Mosaic* to be displayed.

### Ports

**Brick**



Selects the brick to be highlighted.

## 5.7 EvalOnLines

The module takes a *LineSet* and a *LargeDiskData* field as input. The field is evaluated at each vertex of the lineset and the result stored in the lineset.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

LineSet.

**Field** [required]

LargeDiskData.

## 5.8 MosaicToLargeDiskData

The module takes a *Mosaic* containing bricks of overlapping image data and converts them to one *LargeDiskData* object stored on disk.

Press the *Apply* button to start the computation.

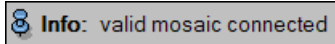
## Connections

### Data [required]

The *Mosaic* object containing the references to the blocks of image data stored on disk.

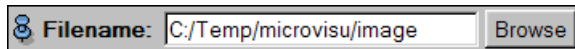
## Ports

### Info



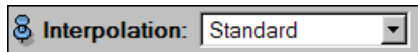
Information on the internal state.

### Filename



The result will be stored in this file.

### Interpolation



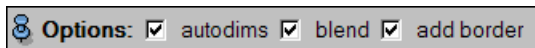
Interpolation method to be used during sampling of the bricks.

*Standard* is trilinear interpolation.

*NearestNeighbor* can be useful if you're working with labels.

*Lanczos* uses a kernel of width 6 to improve the interpolation. The result will be much better but sampling takes much longer.

### Options



Select *autodims* to use the same voxel size as in the input images.

For seamless transitions between bricks, select *blend*.

*add border* extends the result by some amount to guarantee voxels with value 0 at the boundaries. You can specify the border in port *Border*.

### Specify



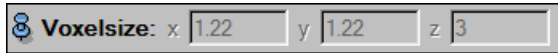
This port is only visible if *autodims* is not selected. It allows you to switch between specifying *dims* or the *voxelsize* of the result.

### Dims



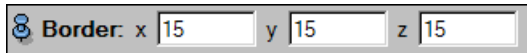
The dimensions of the resulting image. To modify them, *autodims* must be deselected.

### Voxelsize

A control panel for the Voxelsize parameter. It features a blue icon with a white '8' on the left. To its right is the label 'Voxelsize:'. Further right are three input fields: 'x' with the value '1.22', 'y' with the value '1.22', and 'z' with the value '3'. Each input field has a small arrow on its right side.

The voxelsize of the result. You might either specify the dims or the voxelsize.

### Border

A control panel for the Border parameter. It features a blue icon with a white '8' on the left. To its right is the label 'Border:'. Further right are three input fields: 'x' with the value '15', 'y' with the value '15', and 'z' with the value '15'. Each input field has a small arrow on its right side.

Specifies the size of the border to be added on each side of the mosaic. The port is hidden if *add border* is not selected.

## 5.9 Thinner

The module takes labels that have been stored as *LargeDiskData*, and a distance map stored as *LargeDiskData* as input. It runs a thinning procedure on these to extract the center lines of the structure contained in the labels.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

*LargeDiskData* containing labels representing a structure.

**Distmap** [optional]

Distance map to guide the thinning process.

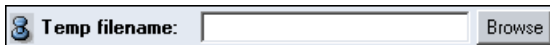
### Ports

**Filename**

A control panel for the Filename parameter. It features a blue icon with a white '8' on the left. To its right is the label 'Filename:'. Further right is a text input field. To the right of the input field is a button labeled 'Browse'.

File the result will be stored in.

**Temp filename**

A control panel for the Temp filename parameter. It features a blue icon with a white '8' on the left. To its right is the label 'Temp filename:'. Further right is a text input field. To the right of the input field is a button labeled 'Browse'.

A temporary file needed by the algorithm.

## 5.10 Threshold

The Threshold module provides simple threshold segmentation.

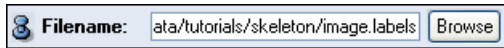
## Connections

**Data** [required]

*LargeDiskData*

## Ports

**Filename**

A graphical user interface element for setting a filename. It consists of a text input field containing the path 'ata/tutorials/skeleton/image.labels' and a 'Browse' button to the right. A small icon is visible on the left side of the input field.

The result will be stored in this file.

**Threshold**

A graphical user interface element for setting a threshold. It features a slider bar with a triangle indicator and a numerical input field to the right containing the value '200'. A small icon is visible on the left side of the slider.

Lets you set the threshold.

## 5.11 TraceLines

The module takes a *LargeDiskData* image as input. This image may be the result of the *Thinner* module. The image should contain only lines represented by voxels. The module traces these lines and builds a lineset and/or a cluster out of it.

Press the *Apply* button to start the computation.

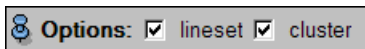
## Connections

**Data** [required]

The image storing the voxel lines.

## Ports

**Options**

A graphical user interface element for selecting options. It shows the word 'Options:' followed by two checkboxes, both of which are checked. The first checkbox is labeled 'lineset' and the second is labeled 'cluster'.

Check *lineset* to enable output of a *LineSet*.

Check *cluster* to enable output of a *Cluster*. The values stored at the vertices indicate the number of neighbors. Endpoints will have a value of 1, vertices a value of 2, and branching points a value of 3 or higher.



## **Part III**

# **Reference Guide - Alphabetic Index of File Formats**



# Chapter 6

## ResolveRT

### 6.1 Analyze 7.5

This format was used by older versions of the Analyze medical imaging software system. Today it has been widely replaced by the *AnalyzeAVW* format. The format is used to store 3D medical images. The actual image data and the header information are stored in two different files. In order to import the data in *amira*, you must select just the header file in the *amira* file browser. The header file is recognized by the extension *.hdr*. If it has some other extension, you must manually select the file format via the file browser's popup menu.

The header file contains the dimensions of the 3D image, the voxel size, and the primitive data type (8/16/32-bit grayscale, 24-bit color). In addition, some other information such as a short data description or a patient id are contained in the header. This information will be stored in the parameter section of the generated *amira* data object.

Note that the input data type may be interpreted as unsigned short if the key for the maximum value (*glmax*) in the header of the *Analyze* file contains a value of larger than 32,768.

### 6.2 AnalyzeAVW

This format is used by newer versions of the Analyze medical imaging software (version 2.5 or higher). Previous versions used the *Analyze 7.5* format, where image data and header information were stored in different files. The *Analyze AVW* format can be used to store 2D, 3D, and 4D medical images, but the 4D time series option is currently not supported in *amira*. Additional attributes stored in the files (like patient name or examination time) are stored in the parameter section of the generated *amira* data objects. *Analyze AVW* files are safely recognized by inspecting the file header. The common file name extension is *.avw*.

Besides *Analyze AVW* image files, sometimes also so-called *Analyze AVW* volume files are used, which contain a list of 2D file names forming a 3D image stack. Such volume files currently cannot be interpreted by *amira*. However, note that *amira* provides a very similar concept with the *Stacked-Slices* format.

*amira* exports scalar and color fields with uniform coordinates to the *Analyze AVW* format. Other coordinate types can be written as well, but in these cases the coordinate information will be lost and no accurate recovery will be possible.

## 6.3 Bio-Rad Confocal Format

The Bio-Rad confocal file format is used to store 3D image data from confocal microscopy. It essentially consists of a 76 byte header section followed by the image data in big endian raw format. *amira* recognizes Bio-Rad files automatically by the suffix *.pic*. In order to load Bio-Rad files from the command line use `load -biorad <filename>`.

Since the header section of the format doesn't contain full information about the voxel size, the bounding box of the 3D image has to be adjusted manually for the resulting uniform scalar field using *amira's crop editor*. Note that Bio-Rad confocal files can only be read but not be written by *amira*.

## 6.4 Leica 3D TIFF

This reader is able to read 3D TIFF files containing a whole stack of 2D images. In particular, the 3D TIFF format is used by newer Leica laser scanning microscopes.

In addition to the image data itself special parameters like pixel size or slice distances may be stored in a 3D TIFF file. If such information is found it will be interpreted in order to create a uniform scalar field of the proper type. However, if no bounding box information is encountered, the *channel conversion dialog* described in the 2D TIFF section will be popped up.

## 6.5 Leica Binary Format (.lei)

This is the Leica binary file format used by Leica laser scanning microscopes. It consists of an *lei* file as well as several TIFF slices. In order to read these files, select only the *lei* file. Parameters like pixel size or slice distance are read from the *lei* file.

## 6.6 Leica Slice Series (.info)

This is the file format used by the older Leica laser scanning microscopes. It consists of an *info* file as well as several raw or TIFF slices, which all must reside in the same directory. In order to read these

files, select only the *info* file. Parameters like pixelsize or slice distance are read from the *info* file. If the file contains colormaps, they will be read, too.

## 6.7 MRC

MRC is a file format for the exchange of electron microscopy data.

With this reader you can import and export files in the MRC file format into amira. Recognized file extensions for the reader are .mrc and .rec.

Note that this reader expects the file header to be in little endian encoding.

## 6.8 Metamorph STK Format

The MetaMorph Stack (STK) file format is used to encode 3D image data, e.g. from confocal microscopy. It is a special version of the TIFF file format. Thus, STK files are indicated as TIFF files in the format column of the file dialog.

STK files can be read just as ordinary 3D TIFF files. The *channel conversion dialog* is popped up, letting the user decide how to proceed with multiple channel images and letting him define the bounding box of the 3D image. Note that size hints stored in the STK file itself are currently not interpreted. Also note that amira can only read but not write STK files.

## 6.9 Stacked-Slices

This file format allows a stack of individual image files to be read, with optional z-values for each slice. The slice distance need not to be constant. The images must be one-channel images in an image format supported by amira (e.g., TIFF). The reader operates on an ASCII description file, which can be written with any editor. Here is an example of a description file:

```
# Amira Stacked Slices

# Directory where image files reside
pathname C:/data/pictures

# Pixel size in x- and y-direction
pixelsize 0.1 0.1

# Image list with z-positions
picture1.tif 10.0
picture7.tif 30.0
```

```
picture13.tif 60.0  
colstars.jpg 330.0  
end
```

Some remarks about the syntax:

- `# Amira Stacked Slices` is an optional header, which allows `amira` to automatically determine the file format.
- `pathname` is optional and can be included in case the pictures are not in the same directory as the description file. A space separates the tag "pathname" from the actual pathname.
- `pixelsize` is optional too. The statement specifies the pixel size in x- and y-direction. The bounding box of the resulting 3D image is set from 0 to `pixelsize*(number_of_pixels-1)`.
- `picture1.tif 10.0` is the name of the slice and its z-value, separated by a blank.
- `end` indicates the end of the description file.
- Comments are indicated by a hashmark character (`#`).

# Index

- abberation, [15](#)
- agarose gel, [15](#)
- AlignBlocks, [49](#)
- Analyze 7.5, [63](#)
- AnalyzeAVW, [63](#)
- ApplyMask, [50](#)
- ApplyTemplateToMosaic, [51](#)
- axial blur, [4](#)
  
- background correction, [42](#)
- batch job, [13](#)
- bead extraction, [14](#)
- BeadExtract, [39](#)
- beads, [15](#)
- Bio-Rad Confocal Format, [64](#)
- black level, [6](#)
- border width, [9](#), [12](#), [44](#)
- boundary artifacts, [5](#)
  
- CCD detector, [42](#)
- ChamferMap, [53](#)
- check point files, [13](#)
- CheckNetwork, [54](#)
- confocal microscope, [4](#), [5](#)
- Convolution, [40](#)
- CorrectZDrop, [41](#)
- coverslip, [15](#)
  
- DataPreprocess, [42](#)
- Deconvolution, [43](#)
- deconvolution
  - blind, [4](#), [11](#)
  - non-blind, [4](#)
  - standard, [6](#)
  
- DisplayMosaic, [56](#)
- DistanceMap, [45](#)
  
- embedding medium, [15](#)
- EvalOnLines, [56](#)
  
- flatfield correction, [42](#)
- Fourier transform, [19](#)
- FourierTransform, [47](#)
  
- immersion medium, [15](#)
- in-plane sampling, [5](#)
- initial estimate, [9](#), [12](#), [44](#)
- intensity attenuation, [7](#)
  
- job dialog, [13](#), [45](#)
  
- Lanczos filter, [8](#)
- landmark set, [39](#)
- Leica 3D TIFF, [64](#)
- Leica Binary Format (.lei), [64](#)
- Leica Slice Series (.info), [64](#)
  
- maximum-likelihood method, [4](#), [43](#)
- memory consumption, [19](#)
- Metamorph STK Format, [65](#)
- microsphere, [15](#), [39](#)
- MosaicToLargeDiskData, [56](#)
- MRC, [65](#)
- multi-processing, [19](#)
  
- noise, [4](#), [6](#), [41](#)
- numerical aperture, [5](#), [44](#)
- Nyquist sampling, [5](#)

- oil immersion, 15
- optical sectioning microscopy, 4
- out-of-focus light, 4, 11
- overrelaxation, 9, 12, 44
- oversampling, 6
  
- performance, 19
- power spectrum, 47
- PSF, 4, 7, 39
  - theoretical, 10
- PSFGen, 48
  
- reampling, 8
- refractive index, 15
- refractive index, 5, 44
  
- sampling rate, 5
- saturation, 6
- scanned volume, 5
- Stacked-Slices, 65
  
- Thinner, 58
- Threshold, 58
- TraceLines, 59
  
- undersampling, 6
  
- vector theory, 48
  
- wavelength, 5, 44
  - emission, 48
  - excitation, 48
- widefield data, 5