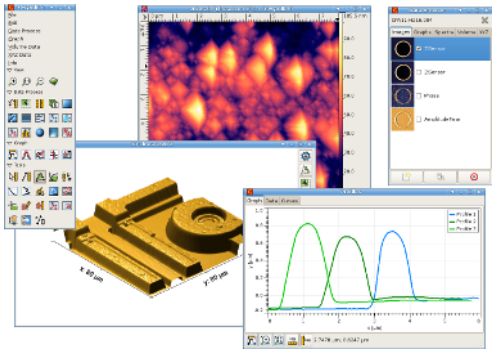
Gwyddion user guide

Petr Klapetek, David Nečas, and Christopher Anderson

This guide is a work in progress.



Contents

Chapter 1

Introduction

This is a user guide and reference for the scanning probe microscopy data processing software Gwyddion.

The latest version of this document is available on-line at https://gwyddion.net/documentation/user-guide-en/. It is available in several languages, namely English, French and Russian. All language versions are listed on-line at https://gwyddion.net/-documentation/.

If you wish to refer to Gwyddion in a scientific paper please cite David Nečas, Petr Klapetek, Gwyddion: An open-source software for SPM data analysis, Cent. Eur. J. Phys. 10(1) (2012) 181-188 (instead or alongside this guide).

1.1 Motivation

Gwyddion is a modular program for SPM data analysis. Primarily it is supposed to be used for analysis of height fields obtained by means of scanning probe microscopy techniques (AFM, MFM, STM, NSOM). However, it can be also used for any other height field and or (greyscale) image processing, for instance for the analysis of profilometry data or thickness maps from imaging spectrophotometry. Gwyddion is Free Software (and Open Source Software), covered by GNU General Public License (GNU GPL).

The main idea behind Gwyddion development is to provide modular program for 2D data analysis that could be easily extended by modules and plug-ins with no need of core recompilation. Moreover, the status of free software enables to provide source codes to developers and users, which makes the further program improvement easier.

Gwyddion can be currently used with Linux/Unix (including Mac OS X) and Microsoft Windows operating systems. Both families of systems can be used also for development. For graphical interface, GTK+ widget toolkit is used, therefore it can be basically ported on any system that is supported by GTK+.

Gwyddion core development is currently funded by Czech Metrology Institute. The project started as a part of the Nanomet initiative (covered by Euromet) in August, 2004. It is supposed that more persons and institutions will participate on development. Project is open for anyone. Welcome...

1.2 Licensing

Gwyddion is covered by GNU General Public License (GNU GPL). The full license text is also included as file `COPYING` in the source distribution (MS Windows installers contain it as file `COPYING.wri`). In brief, this license means that:

- You can freely use the program. You can freely make copies, modify and distribute them. You can download the program and its source code from Gwyddion web pages and modify it as you want.
- If you decide to distribute it, the modified code is still covered by the same license. In particular, you have to offer the source code too.
- The same holds for extensions, e.g. if you write an import module for a new file type or a new data analysis function it has to be licensed under GNU GPL (when it is distributed – if you use it privately, do whatever you wish with it).
  However, it is also possible to execute third-party programs from Gwyddion and these do not necessarily have to be distributed under the same license – if they are not derived works of Gwyddion (which, admittedly, is not always easy to determine).

The main reasons why the program is covered by this kind of license are here: first of all, this licensing policy enables us to make modular program that can be easily developed by many people from different institutions. Second, this license protects the rights of developers that their code, here given to the public, cannot be copied and used for closed proprietary products.

Chapter 2

Installation

Gwyddion source code and executables can be downloaded from the download web page of the project, or alternatively from raw SourceForge.net download page. The installation varies depending on the operating system and the various installation methods will be described in the following sections.

To play with Gwyddion you might also want to download the sample Gwyddion files. They are in native Gwyddion format and represent typical AFM data.

2.1 Linux/Unix Packages

Some GNU/Linux and Unix systems provide binary packages of Gwyddion. The download page of the project also tracks known packages and packaging efforts. For instance, Debian, Ubuntu, Gentoo, openSuSE or FreeBSD offer Gwyddion packages. A Flatpak app installable in multiple distributions also exists. Gwyddion is available in several Mac OS X port/package repositories, such as MacPorts, Homebrew and Fink. If your operating system provides such a package and it is recent enough, install it using the standard means of the operating system. Otherwise you may need to proceed with compilation from source code.

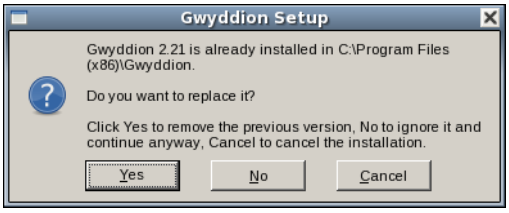
2.2 MS Windows Packages

MS Windows packages built by the developers are available on the project download page. At present both 32bit and 64bit executables are provided. Most new users should choose the 64bit package package nowadays. The 32bit package still has some uses though.

Note MS Windows will warn the program comes from an unknown publisher, is unsafe, not trusted or something along these lines (the wording depends on OS version). This is because in order to avoid the warning we would have to pay regularly hundreds of € to some certificate company. If you downloaded the installer from SourceForge it is the official Gwyddion installer and there will not be any you can trust more. You simply have to proceed despite MS Windows attempts to discourage you.

Note The packaging of MS Windows executables has changed substantially in version 2.23. So, if you upgrade from a pre-2.23 version to version 2.23 or newer, please read the description of the changes.

If you have already installed Gwyddion the installer asks if you want to replace the previous version.



The question what to do with the previous Gwyddion version.

You have three options:

Yes (replace) The already installed version will be replaced. This is the normal upgrade method.

No (keep) The already installed version will be ignored and the installation will proceed as a fresh one. Generally, this is a bad idea as both versions will share settings and registry keys and if you uninstall one the other will be affected. Furthermore, it does not remove any of the files of the already present version – and they can get into the way. Sill, you may find this option useful in some cases.

Cancel The installation will be aborted and the old version will be kept untouched.

In the following steps the installer reminds you of the software components included in the package and their licenses (that are all Free Software), lets you change the installation directory and offers a choice of languages to use for the user interface.

Uninstallation

If you want to uninstall Gwyddion go to *Start* → *Control Panel* → *Add or Remove Programs* and choose Gwyddion. Note that this is valid for Windows XP. The path to the *Add/Remove* window may be slightly different on other Windows OS.

Registry keys

The installer creates the following useful keys under `HKEY_LOCAL_MACHINE\Software\Gwyddion\2.0`:

`InstallDir` Installation directory, e.g. `C:\Program Files\Gwyddion`. Reading this key can be useful for determining where to install extensions.

`Version` Full Gwyddion version as a string.

`Locale` Language of Gwyddion user interface chosen during the installation (more precisely, a locale specification that, among other things, defines the language). You can modify it using **regedit** to choose another language as described below.

The list of available languages and corresponding `Locale` values include:

`Locale`	Language
`cs_CZ.UTF-8`	Czech (Czech Republic)
`en_US.UTF-8`	English (United States)
`en_GB.UTF-8`	English (United Kingdom)
`fr_FR.UTF-8`	French (France)
`de_DE.UTF-8`	German (Germany)
`it_IT.UTF-8`	Italian (Italy)
`ko_KR.UTF-8`	Korean (South Korea)
`ru_RU.UTF-8`	Russian (Russia)
`pt_BR.UTF-8`	Portugese (Brazil)
`es_ES.UTF-8`	Spanish (Spain)

Missing features

Gwyddion has a large number of optional features that depend on third party libraries. The MS Windows packages contain most of them but a few are not included at present:

- Flexible Image Transport System (FITS) file import.
- Pygwy support in the 64bit packages (it is supported only in the 32bit packages).

Enabling pygwy

The Python scripting interface, pygwy, is included in the installer. However, you need to install Python and PyGTK2 separately to use Python scripting. This can be done either prior to Gwyddion installation or any time later. If Python and PyGTK2 is not present pygwy simply does not register itself upon Gwyddion startup.

MS Windows Python installer could be obtained at https://www.python.org/downloads/. Since pygwy requires Python 2 you need a Python 2.7 package, which are no longer officially supported. The latest version `python-2.7.16.msi` is mirrored at Gwyddion's SourceForge page in case you have trouble finding or downloading it.

Three packages are required for PyGTK2: PyGTK, PyCairo and PyGObject. Follow the corresponding download links for these modules at https://www.pygtk.org/downloads.html to obtain the installers `pygobject-2.28.3.win32-py2.7.msi`, `pycairo-1.8.10.win32-py2.7.msi`, and `pygtk-2.24.0.win32-py2.7.msi` or possibly newer versions (if available). They also mirrored at Gwyddion's SourceForge page in case you have trouble finding or downloading them.

Success has also been reported with the all-in-one installer `pygtk-all-in-one-2.24.2.win32-py2.7.msi` that contains everything. However, using the all-in-one installer means entire GTK+ will be installed twice (into different locations). Which bits of which installation will be used in pygwy is difficult to tell. Hence this method is not recommended.

Chapter 3

Getting Started

This chapter introduces various basic concepts and terms, such as masks or selections, explains the organization of data in Gwyddion and describes the user interface.

The descriptions are relatively thorough and some of the topics near the end of the chapter, such as raw file import, might be considered advanced and not for everyday use. So, despite the name, it is not necessary to read this entire chapter to be able to work with Gwyddion. The Gwyddion user interface is intuitive and much can be discovered by playing. The clarification of the basic ideas and components provided here will hopefully ease the discovering.

You can also get help for the current function or window from within Gwyddion. Pressing **F1** or clicking on the *Help* button will in most windows show a relevant part of the on-line version of this guide in a web browser. The starting page of guide can be displayed by *Info* → *User Guide*. Of course, this works if you are on-line and Gwyddion finds a suitable web browser. See the settings tweaks if you want or need to tweak the setup, e.g. for an off-line version of the guide.

Tip Command *Info* → *Tip of the Day* displays data processing tips and highlights useful features that you might miss.

3.1 Main Window

The main window, also called toolbox, is one of the two Gwyddion windows that appear after program start (with no files given to open), the other is the data browser. Closing the main window causes Gwyddion to exit.

The toolbox contains the set of Gwyddion menus and from several rows of buttons connected with common functions and tools. The menus group the functions as follows:

File associates commands that are used for file loading and saving. Certain global commands (e. g. *Exit*) are located here too. The history of recently opened files can be browsed with *File* → *Open Recent* → *Document History*.

Edit provides history manipulation commands (*Undo*, *Redo*) and editors of miscellaneous global resources, such as gradients and materials for false colour and 3D data representation or the default colour used for data masking.

Data Process is built automatically from all data processing modules available in the Gwyddion module directory (depending on the operating system). This menu together with *Tools* panel of buttons contain most of the commands you will need at analysing your SPM data. A subset of these functions is also available in *Data Process* button panel. These buttons serve as shortcuts to commonly used functions from *Data Process* menu. All functions accessible from *Data Process* button panel can be found in the menu too.

Graph is similar to *Data Process*, except it consists of graph functions. Graph processing includes function fitting, exporting graph data etc. Button panel *Graph* again contains a subset of the commonly used functions from *Graph* menu.

Volume Data is similar to *Data Process*, except it consists of volume data functions.

XYZ Data is similar to *Data Process*, except it consists of XYZ data functions.

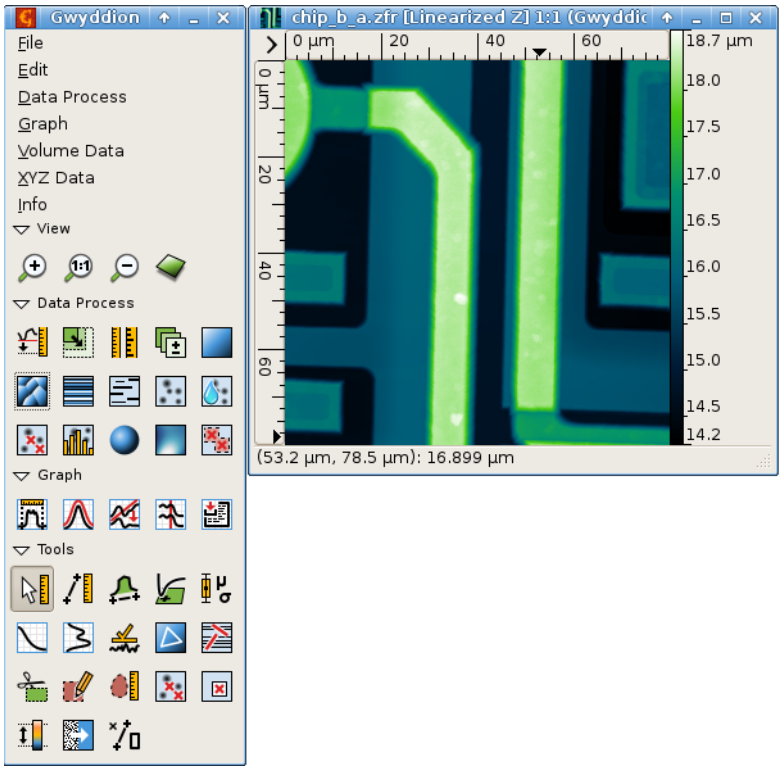
Curve Maps is similar to *Data Process*, except it consists of curve map data functions.

Info contains commands that provide various auxiliary information about Gwyddion, such as program version or the list of loaded modules and their functions.

Finally, you can find some rows of buttons in the main window. Buttons in *View* panel offer zooming functions (that are often more easily invoked by keyboard shortcuts or just resizing the data window) and 3D data display. Panels *Data Process* and *Graph* contain selected functions from *Data Process* and *Graph* menus as described above.

Panel *Tools* contains tools, i.e. functions that directly work with selections on data windows. These functions are accessible only from this button panel.

The toolbox button groups can be modified using *Edit* → *Toolbox*, including the complete removal of some groups and creation of different ones.



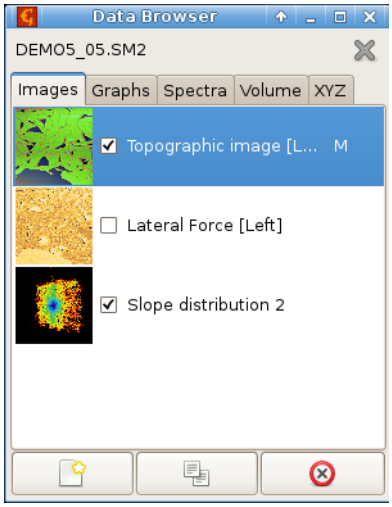
Main window and a data window showing microchip surface (Gwyddion sample file `chip.gwy`).

3.2 Data Browser

Data browser is a window that displays the structure of currently focused file. It shows the content as represented in Gwyddion which may differ somewhat from the organization in the original software.

Gwyddion supports an arbitrary number of two-dimensional data fields per file. Depending on the context, they are also often called images or height fields in this guide (individual images are also occasionally called channels although other, non-image data types consist of channels as well). The dimensions of images in one file may differ and also the physical dimensions and values can be arbitrary physical quantities.

In addition, one-dimensional data, represented as graphs, and single-point spectra can be present in the same file. The data browser is a tool for browsing and managing all the available data in the file.



Data browser displaying several images.

Controlling the Browser

Since the data browser always displays the structure of currently focused file, its contents change as you switch between different windows, possibly showing data from different files. There is no difference between native Gwyddion files (`.gwy`) and other files. Once a file is loaded its structure is shown as if it was a Gwyddion file.

The data browser has several tabs, one for each type of data that can be present in the file:

- *Images*
- *Graphs*
- *Spectra*
- *Volume*
- *XYZ*
- *Curve Maps*

Each list shows names of the data objects and some additional properties that depend on the specific data type. The names can be edited after double-clicking on them.

Individual images, graphs, spectra, volume, XYZ or curve map data can be deleted, duplicated or extracted to new Gwyddion native file using the buttons at the bottom of the browser. It is also possible to copy them to another file by dragging a data browser row to any window belonging to the target file. To rename a data item, select it and press **Enter** or triple-click on it with the mouse.

The close button in the top right corner of the data browser closes the current file, discarding all unsaved changes. A file is also closed when all windows displaying data from this file are closed.

If the data browser is closed it can be recalled using the *Info* → *Show Data Browser* command.

Images

The image list shows thumbnails, check-boxes controlling whether the image is visible (i.e. displayed in a window) and image names. Right to the name the presence of presentation, mask or calibration is indicated by the following letters:

- *M* – mask
- *P* – presentation
- *C* – calibration

Graphs

The graph list shows check-boxes controlling whether the graph is visible and graph names. Right to the name the number of curves in the graph is displayed.

Spectra

The spectrum list shows the spectra name and the number of points in the set. Since single-point spectra are displayed and operated on only in connection with a two-dimensional data using the spectroscopy tool, there is no check-box controlling the visibility.

Volume

The volume data list shows the data name and the number of levels in the z direction, i.e. perpendicular to the section displayed in the window. A letter *Z* after the number of levels indicates the presence of a z-axis calibration.

XYZ

The XYZ data list shows the data name and the number of points in the XYZ point set.

Curve Maps

The XYZ data list shows the data name and the number of curves (data series) in each point of the map. If the curves are segmented the number of segments is displayed after a colon.

3.3 Managing Files

Gwyddion uses its custom data format (`.gwy`) to store data. This format has the following important advantages:

- Capability to store the complete state of the individual data, including masks, selections and other properties.
- Arbitrary number of images, graphs, spectra, volume and XYZ data sets, with arbitrary dimensions and units of both independent variables and values.
- Double-precision representation of all data, preventing information loss due to rounding.

Therefore, we recommend to use this format for saving of processed files.

Other data file formats are handled with appropriate file loading and saving modules. Beside a large number of file formats used in scanning probe microscopy, graphical file types (PNG, JPEG, TIFF, TARGA) and raw binary and text data can be imported too. If your SPM data format is not supported by Gwyddion yet or it is loaded incorrectly, you are encouraged to write an import module (if you can program) or contact the maintainers to help them improve the support.

The list of all supported file formats can be found in chapter Summaries and Tables.

File Loading

Files are opened using *File* → *Open*. The file type is detected automatically, based solely on the file content. Since the same extensions such as `.img`, `.afm` or `.dat` are used for many different SPM file types this approach is superior to relying on file extensions.

The only exception is the import of various raw data, either two-dimensional or graph, that must be chosen explicitly in the file open dialogue. See sections Raw Data File Import for details of import of raw data and manual extraction of data from unsupported formats and Specific Data Import for import of XYZ data, pixmap image data and graph data.

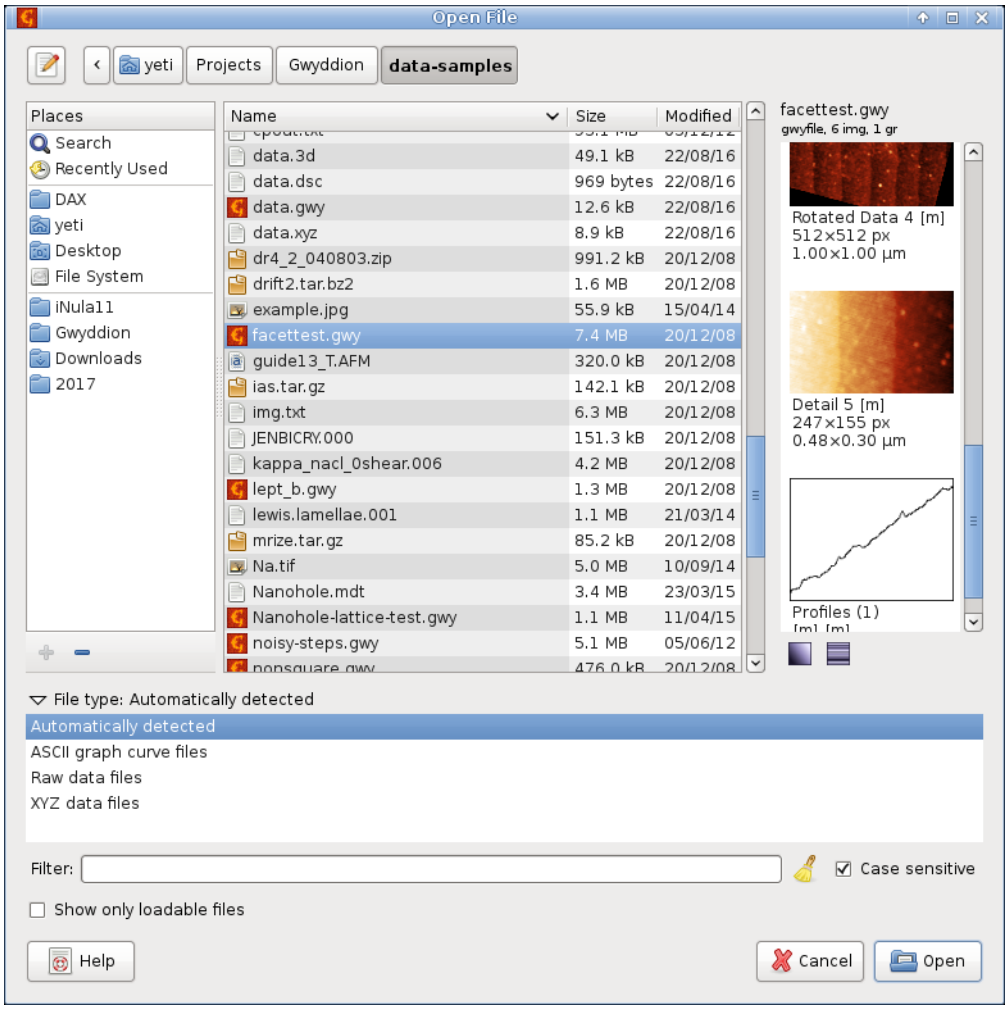
The list of files in the file open dialogue can be limited to only files Gwyddion recognizes as loadable by enabling the *Show only loadable files* option. The file type label then indicates the filtering by appending *(filtered)* to the end. This can be often convenient, on the other hand it can slow down listing of directories with many files.

The file list can be also filtered by name using the *Filter* entry in the lower part. You can enter either a file glob such as `*.dat` or a file name fragment such as `carbo`. Press **Enter** to update the list after modifying the filter.

Note Filters and raw import options are remembered. If you open the file chooser after a while and files seem to be missing or suddenly not loadable, check the filtering and file type settings.

A preview of the file content is show on the right side (for automatically loadable files). The small text above the preview shows the file type, i.e. name of the module used to load the file, and the number of images (img), graphs (gr), single-point spectra (sp), volume data (vol) and XYZ data (xyz) in the file.

Tip The preview of image channels can display the data be plane-levelled and/or row-corrected. This is controlled by the small buttons below the preview list.



File open dialogue with expanded file type options and file content preview.

File Merging

File merging, performed by *File* → *Merge*, is similar to normal file loading, except that the selected file (or files) is merged into the current open file. In other words, images, graphs and other types of data are added to those already present in the current file, together with all their settings and properties.

File Saving

Much of the previous paragraphs applies to file saving too. One of the main differences is the reliability of automatic file type determination. While loading can and does examine the file contents, saving depends on file name and extension. Combined with the large number of different file types using the same extension such as `.img`, `.afm` or `.dat` it leads to ambiguities. Select the file type explicitly before saving if you are unsure.

Since the only file type able to fully represent Gwyddion data structures is its native data format, saving to a `.gwy` file is the only proper saving. Saving to other file formats essentially consists of exporting of a limited subset of the data, typically only the active image (without masks and presentations). Therefore it does *not* change the file name of the current file to the just saved file name.

File → *Save as...* can also be used to export image data to graphics formats. Just enter `foo.png` as the file name to export a PNG image the current channel, similarly for other formats.

Document History

The history of recently opened files can be accessed with *File* → *Open Recent*. The submenu contains the last 10 recently used files for quick recalling, an extensive recent file history is accessed with the last item *Document History*.

Document history lists the files sorted by the last access time (the most recently accessed at the top), with previews and some additional information about a selected channel. The function of the bottom row of buttons is following:

Clean Up Removes history entries of files that have been deleted or are no longer accessible for other reasons.

Close Closes the document history window.

Open Opens the selected file. This can be also achieved by activating the selected row, either by double-clicking or with the keyboard.

The history can be searched/filtered by file name using the filter controls above the buttons. The filter is activated by pressing **Enter** in the filter pattern entry. To display all history entries, clear the entry and activate it. The filter pattern is interpreted in two ways:

- If the pattern contains wildcards, i.e. `*` or `?`, it is interpreted as file glob. This means `?` represents a single arbitrary character, `*` represents an arbitrary sequence of zero or more characters, and the file name has to precisely match the pattern. Note directory separators (`/` or `\`) are not treated specially, therefore in the pattern `*.sis` the initial `*` matches all leading directory components. The pattern syntax is described in GPatternSpec documentation.
- If the pattern does not contain any wildcards, it is directly searched as a part of the file name.

Search case sensitivity, controlled by option *Case sensitive*, is useful mainly on systems distinguishing letter case in file names, such as Unix. On systems that do not distinguish the case themselves it is recommended to keep the setting on case insensitive.

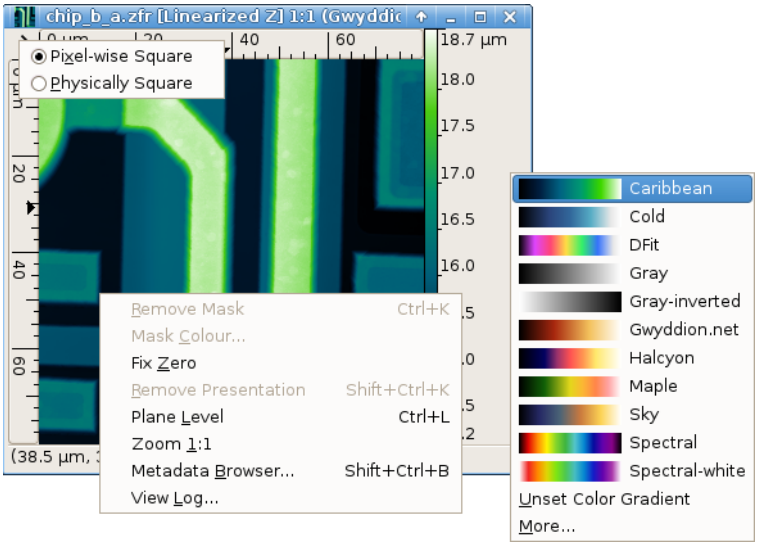
3.4 Data Window

Two-dimensional data are presented in so called data windows. It is the main widget used for working with Gwyddion. The data are presented as a field of false colours corresponding to heights. Colour axis that represents mapping of colours to real height values is on the right side of the data window.

The False colour palette used to represent height data can be changed by clicking on the colour axis with right mouse button (i.e. invoking context menu) and selecting a palette from the list. Most frequently used palettes are available directly in the context menu; however you can reach much more of the possible palettes using the *More* menu entry. Moreover, you can use the colour gradient editor to create your own palettes and select which palettes should be displayed in the short list.

There is a context menu available also for the data area. This menu consists of basic data and presentation operations. To reach all the possible operations use *Data process...* menu at the Gwyddion main window, or use some of the tools available at the *Tools* set of buttons at the Gwyddion main window.

The arrow in the upper left corner brings the aspect ratio switch menu. The data can be displayed either with pixels mapped with 1:1 aspect ratio to screen pixels (*Pixelwise Square*), or with physical dimensions mapped 1:1 onto the screen (*Physically Square*). For instance a sample of size 1×1 μm scanned with reduced slow axis resolution and having therefore 512×128 pixels, can be displayed either in 512×128 or 512×512 window.

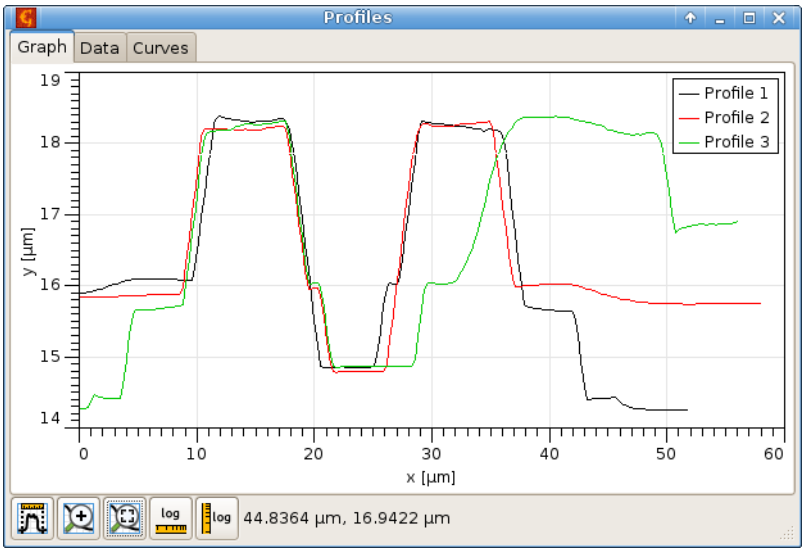


Data window with all three available context menus shown.

3.5 Graph Window

Graph window is used for one-dimensional data processing. They are created by appropriate tools or modules that extract graphs from height field data.

Graph window consist of three tabs: the first two represent graphical and tabular views of the 1D data and the third one shows a list of graph curves. Several tools connected with viewing 1D data are available directly in the graph window toolbar, namely zoom buttons and logarithmic axis buttons. To reach all the possible operations use *Graph* menu in the Gwyddion main window or click with right mouse button inside the graph area.



A graph window with three profiles.

To edit curve presentation one can either click on the curve in the graph or activate (double-click) the corresponding row in *Curves*. Individual curves can be deleted by selecting them in *Curves* and pressing **Delete**. It is also possible to copy individual curves to other graphs by dragging their curve list rows onto them (provided the graphs are unit-wise compatible).

Clicking on a graph axis brings a dialog with axis properties and graph key properties can be edited after double-clicking on it.

3.6 Tools

Functions from *Data Process* menu and button panel either execute immediately or after asking for parameters and settings in a dialog box. Tools, accessible from *Tools* button panel, work differently. Once selected, they remain active and always follow the current data window until one switches to another tool. In other words one can switch data windows freely while using a tool and it always shows information from or operates on the current data. Tools also differ by working with selections, e.g. points, lines or rectangles, on data windows. Nevertheless functionally they perform similar tasks as *Data Process* functions – value reading, leveling, statistics, correction, etc.

Tools can be launched only from *Tools* button panel located in the main window. Gwyddion includes these tools:

Read Value Reads values at the position of mouse click.

Distance Measures distances – similarly to Read value this tool enables user to measure horizontal, vertical and Euclidean distance and angle between points in the data field. In addition it displays the difference of data values between points.

Profiles Along Axes Extracts horizontal and/or vertical scan line profiles from the image and puts them to separate graphs. These graphs can be further processed with commands from the *Graph* menu.

Profile Extracts profiles along arbitrary lines from the image and puts them to separate graphs. These graphs can be further processed with commands from the *Graph* menu.

Radial Profiles Extracts radial (angularly averaged) profiles from the image and puts them to separate graphs. These graphs can be further processed with commands from the *Graph* menu.

Spectro Views and extracts single point spectroscopy data.

Statistical Quantities  Computes basic statistical quantities (RMS, Ra, minimum, maximum, projected and surface area, etc.) from a selection of full data field. It can also calculate them only on the masked area, or even combine these two types of selection of area of interest.

Statistical Functions  Computes basic statistical functions (distribution of heights or slopes, autocorrelation function, power spectrum density function, etc.) from a selection of full data field.

Row/Column Statistics  Somewhat complementary to 1D statistical functions, this tool plots characteristics such as mean, median or surface length for each row (column).

Correlation Length  Quick estimation of autocorrelation length T from scan lines using several common methods.

Roughness  Evaluates standardized one-dimensional roughness parameters.

Three Point Level  Levels data by plane obtained by clicking on three points within data window. The three values can be averaged over a small area around the selected point.

Path Level  Row leveling tool equalizing the height along a set of arbitrary straight lines.

Crop  Cuts part of the data.

Mask Editor  Manual editing of masks: creation, exclusion, intersection, inversion, growing and shrinking, ...

Grain Measurement  Measures individual grain parameters.

Grain Remover  Removes continuous parts of the mask by clicking on mask point and/or interpolates (removes) data under a continuous part of mask.

Spot Remover  Manually removes spots. Select a point on a data window, mark an area to interpolate on the zoomed view and remove the defect using chosen interpolation method.

Color Range  Stretches color range or changes false color mapping type. It enables the user to change the false color representation range (by default from data minimum to data maximum).

Filter  Basic filters – mean, median, conservative denoise, minimum, maximum and similar simple filters to reduce noise in the data.

Selection Manager  Displays selections for an image channel and copies them to other images or files.

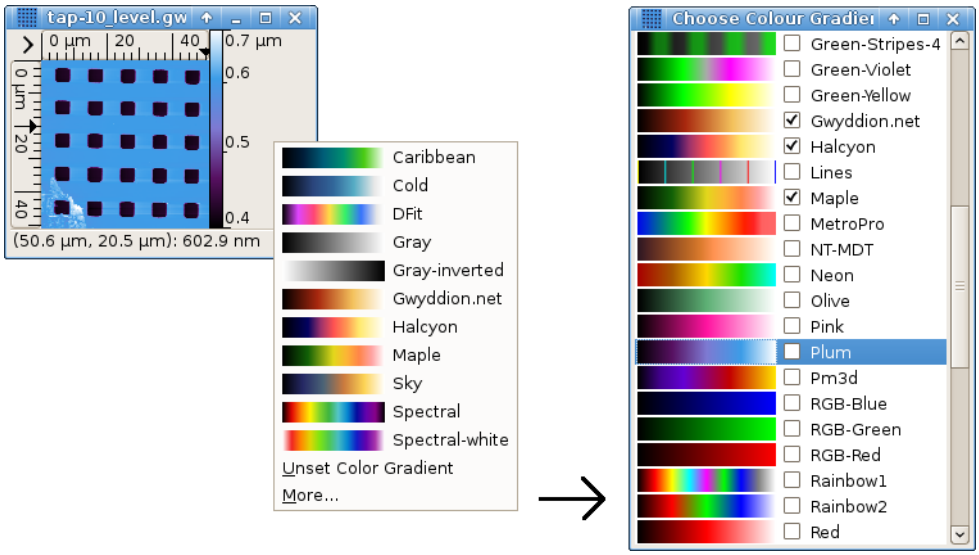
Tool dialogs can be closed (or more precisely hidden, as the current tool is still active even if its dialog is not visible), beside activating the *Hide* button, by pressing **Esc** or clicking the tool's button in the toolbox again.

3.7 False Color Mapping

False color mapping is the basic two-dimensional data visualization method. The color gradient (also called palette) to use can be selected after clicking on the false color map part of a data window with right mouse button.

This quick selection pop-up menu offers the list of preferred color gradients. In addition it allows to invoke the full color gradient list by selecting *More*. Preferred gradients can be chosen by checking the corresponding check buttons in the full list or in the gradient editor list. Selecting a row in the full list changes the gradient to the selected one, double-clicking (or pressing **Enter**) also finishes selection and closes the list window. Gradients of known names can be quickly accessed by starting to type their name. The default color gradient to use (when none is specified in the file) can be also set in the gradient editor.

More control over the way values are mapped to colors is possible with Color range tool.



A data window with the right-click color gradient pop up menu and the full color gradient list.

Color Range Tool 

The Color range tool is a special tool whose purpose is not to analyse or modify data, but to control the way values are mapped to colors. It offers four basic color mapping types:

Full Data values are mapped to colors linearly, the full data range corresponds to the full color range. This is the default type (unless you have changed the default).

Fixed  Data values are mapped to colors linearly, a user-specified data range (which can be smaller or greater than the full range) maps onto the full color range. Values outside this range are displayed with the edge colors. The range can be set by several means:

- by entering desired values numerically in the tool window,
- by selecting a range on the height distribution graph in the tool window,
- by selecting an area on the data window, the range is then set from the minimum to maximum of the data in this area only,
- by pressing buttons *Set to Masked* or *Set to Unmasked* that set the range to the range of values that are under or not under the mask, respectively, or
- by pressing button *Invert Mapping* which exchanges the upper and lower limit of the color mapping range.

If no range is manually set, fixed range type behaves identically to full range.

Note data processing operations often modify the value range – and as the fixed range remains fixed as you set it, it can result for instance in completely black data display. You may wish or have to update the range manually then, or to switch to another mapping type.

Automatic  Data values are mapped to colors linearly, a heuristically determined subinterval of the full value range maps onto the full color range. Values outside this subrange are again displayed with the edge colors.

Adaptive  The full data range corresponds to the full color range, however data values are mapped to colors non-linearly. The mapping function is based on inverse cumulative height distribution, therefore flat areas generally get bigger slice of the color gradient and smaller value variations can be seen on them than normally.

The false color map ruler on the right side of data windows does not display any ticks in this mode, only the minimum and maximum value.

A mapping type can be set to be default by checking the *Default* check button when it is active. Newly displayed data windows then use this type, unless some other type is explicitly specified.

Saving data to `.gwy` file also saves all color mapping settings: mapping type, range and gradient. Gradient is however not physically stored in the file, only referenced by name. In other words, color gradients of the same name are shared among files.

Color Gradient Editor

Color gradient editor can be invoked with *Edit* → *Color Gradients*. It consists of a gradient list similar to the full gradient selector, with an additional button panel, and the actual editor that can be invoked by double-clicking on a gradient you wish to edit or by activating the *Edit* button. Renaming is possible. Only user-created color gradients can be edited or deleted, system gradients installed with Gwyddion are immutable.

The last button in the gradient list control panel makes the currently selected gradient the default. It will be used for all newly displayed data that do not specify any particular color gradient.

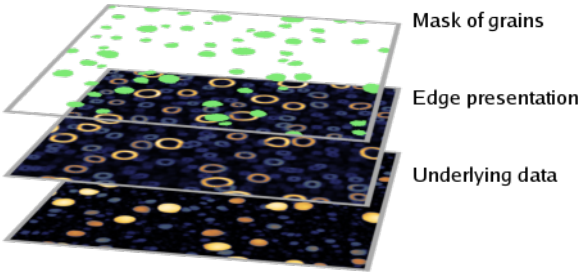
Two editing modes are available:

Points The color gradient is defined by a set of points and their associated colors. The points are represented by triangular markers on the gradient displayed in the lower part of the editor window. Moving these markers moves the points, new points can be added by clicking into an empty space, existing points can be removed by dragging them away from the gradient.

Curve The color gradient is defined by red, green and blue curves. The curves are again segmented, but the segments of individual curves do not need to coincide.

3.8 Presentations and Masks

Presentations and masks are secondary images with specific functions attached to the data. Presentation is a different image displayed over the actual data. Mask represents an irregular set of selected pixels. Their presence is indicated by the flags *P* (presentation) and *M* (mask) in the data browser.



Visualization of masks and presentations. If you look from above they can be imagined to be stacked as in the picture.

Both masks and presentations can be removed from the data by functions in the right-click menu of the data window, or with keyboard shortcuts.

Data in default false colour representation (left), with superimposed mask visualized with a red colour (centre) and with shading presentation (right).

Presentations

Presentations can be used to show the height field in another way than as a false colour map of the heights, for instance with shading or with highlighted edges. It is also possible to superimpose an arbitrary data field over another one as the presentation.

Note the superimposed presentation is really only a presentation, it is never used for calculations. In all data processing functions or tools the results are always computed from the original underlying data. Since presentations can be computationally intensive to calculate, they are not automatically updated when the underlying data change. The various presentations available are described in section Presentations.

Since the values of the presentation image have arbitrary absolute scale and may not even correspond to any meaningful physical quantity, the false colour map ruler on the right side of the data window does not display any ticks nor the minimum and maximum value when a presentation is shown.

Masks

Masks are used for special areal selections, e.g. grains, defects or facets with certain orientation. Masks can have any shape and within the data window and they are visualized by a colour overlaid over the data. The mask colour and opacity can be changed in the right-click context menu of the data window.

Many functions, especially statistical, levelling and row correction functions, can use masks to include or exclude areas from the calculation. This choice is presented as *Masking Mode* with the options to exclude the masked region, include only the masked region or ignore the mask and use the entire data. Note that this choice is usually displayed only if a mask is present.

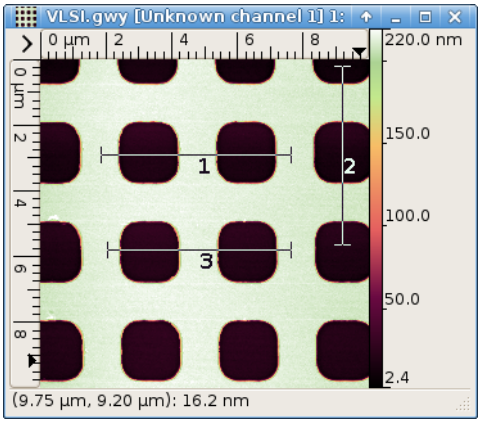
Since grain marking is the most common use of masks, several functions that operate on marked areas are called "grain" functions, e.g. Grain Statistics. Also, a contiguous part of mask is sometimes called grain in this guide. However, since a mask does not bear any information how it was created all mask functions can be used with masks of any origin.

3.9 Selections

All interactive tools and some other processing methods allow to select geometrical shapes on data with mouse: points, lines, rectangles, circles/ellipses. Existing selections can be similarly modified by dragging corners, endpoints, or complete selections. When mouse cursor is moved near to an editable point of a selection, is changes its shape to indicate the possibility to edit this point.

Each tool typically uses only one type of selection and when it is activated on a data window, it sets the selection mode to this type. Selections of other types than currently displayed are remembered and they are recalled when a tool which uses them is activated again. E.g. when you select several lines with Profile extraction tool, then switch to Statistical quantities (the lines disappear) and select a rectangular area to calculate its statistical characteristics, and then switch back to Profile extraction, the rectangle disappears and the lines appear again.

Tools that use the same type of selection – e.g. both Statistical functions and Statistical quantities use rectangular selection – share it. To calculate height distribution of the same rectangle you have selected for statistical quantities, it is sufficient to switch the tool.



Data window with three selected lines, two horizontal and one vertical.

If you save data in Gwyddion native file format (`.gwy`), all selections are saved together with data and recalled the next time the file is opened and appropriate tool chosen.

Pressing **Shift** during selection restricts the degrees of freedom of the shape, making it easier to draw shapes from a specific subset. Specifically, pressing **Shift** restricts

- rectangular selections to perfect squares,
- elliptical selections to perfect circles, and
- directions of line selections to multiples of 15°.

The last modified shape can also be adjusted using the keyboard:

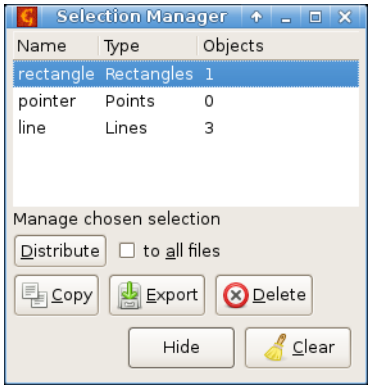
- arrow keys **Left**, **Right**, **Up** and **Down** move the point – for lines or rectangles this means the first point,
- holding **Shift** moves the other point instead for lines or rectangles, and
- holding another modifier key such as **Alt** or **Ctrl** moves the point by larger steps.

Several tools can hold the selection while switching between images. Meaning the selection is copied from the current image any image you switch to. It is enabled by checking *Hold selection* in the tool. Depending on the additional options, the current selection can be copied only to images which do not have any yet or replace existing selections.

Selection Manager

The selection manager tool is a special tool that displays the list of all selections in an image and enables to copy them to other images.

For each selection, the tool shows the name, which is how the selection is identified in the `.gwy` file; the selection type and the number of objects (points, lines, rectangles, ...) selected. Usually, there is at most one selection of any type because they are shared among the tools as described above. Nevertheless, sometimes there are special or private selections present as shown on the following figure displaying two point-wise selections.



Selection Manager showing several selections present in the data.

Selection chosen in the list is displayed in the data window.

It is possible to delete individual selections by choosing them in the list and pressing the **Delete** key or *Delete* button – this is equivalent to clearing the selection in the corresponding tool. The *Clear* button removes all selections.

However, the most interesting function of Selection Manager is selection copying. There are two ways to copy a selection to another image:

- Dragging a row from the selection list onto a data window copies the selection to this data window.
- Clicking the *Distribute* button copies the selection to all other images in the file. Or, if *to all files* is enabled, to all images in all open files.

Selections are copied only to images with compatible lateral units. This means that a selection in a normal image with meters as the lateral units will not be distributed to a two-dimensional PSDF image or a two-dimensional slope distribution.

If the physical dimensions of the target data are not sufficient to contain all the objects of the copied selection then only those objects that fit are copied (this can also mean nothing is copied).

The coordinates defining the chosen selection can be exported to the clipboard or a text file using *Copy* and *Export* buttons. Each specific selection type is defined by its coordinates as described in the developer documentation: for instance points have two coordinates x and y, while rectangles have four that correspond to x and y of its two corners. The coordinates are always relative to top left image corner, even if the origin is not in the top left corner.

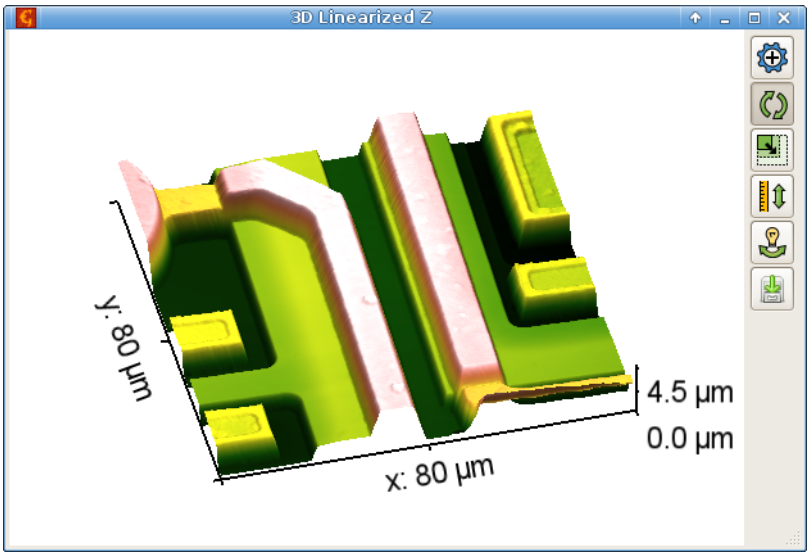
3.10 OpenGL 3D Data Display

Three-dimensional OpenGL display of the current data window can be invoked with the button with symbol of cube in *View* button row of main window.

This feature is optional, i.e. it can be disabled at compile time. It can also happen that while Gwyddion is capable of 3D data display your system does not support it. In both cases an attempt to invoke 3D view gives an error message explaining which

of the two cases occurred. In the former case you have to ask the producers of Gwyddion executables to build them with 3D support or build Gwyddion yourself from source code. If it is the latter case, refer to your operating system guide on how to enable OpenGL 3D capabilities. If you experience a poor performance of the 3D view in MS Windows try disabling desktop composition for Gwyddion (in *Compatibility* tab of the Gwyddion shortcut).

The 3D window has two possible forms: with basic and expanded controls. It starts with basic controls only, this form is displayed on the following figure. It can be switched to the expanded form (and back) with an expander button in the upper right corner. Clicking on the view with right mouse button brings a quick colour gradient/GL material selector.



Three-dimensional OpenGL data display window with basic controls.

Basic Controls

Basic 3D window contains interaction mode controls at the right side of the view. By default, dragging the view with mouse rotates it horizontally and vertically. All possible modes are listed below, together with hotkeys that switch between them:

Rotation (R) The default when the 3D view is newly displayed. Dragging the view horizontally rotates it around z-axis, vertical drag rotates it around horizontal axis parallel with the plane of view.

Scaling (S) Dragging the view upwards enlarges it; dragging in downwards makes it smaller. The size can be also adjusted using the mouse scroll wheel.

Z-scaling (V – value scale) Dragging the view upwards increases the z-scale, making the hills and valleys more pronounced. Dragging it downwards decreases the value scale. The value scale can be also adjusted using the mouse scroll wheel with **Shift** pressed.

Light rotation (L) This control is available when there the visualization mode has a light source, i.e. in lighting and overlay modes. Dragging the view changes position of the light source similarly to rotation of data in normal rotation mode. Horizontal movement changes the φ angle (direction in the horizontal plane). Vertical movement changes the ϑ angle between the light direction and the horizontal plane. The light position is visualised using a "wheel" around the surface.

The basic controls also include an image export button.

Full Controls

In expanded controls the mode buttons are located in top row, however their function does not change. In addition, there are several tabs with options below them:

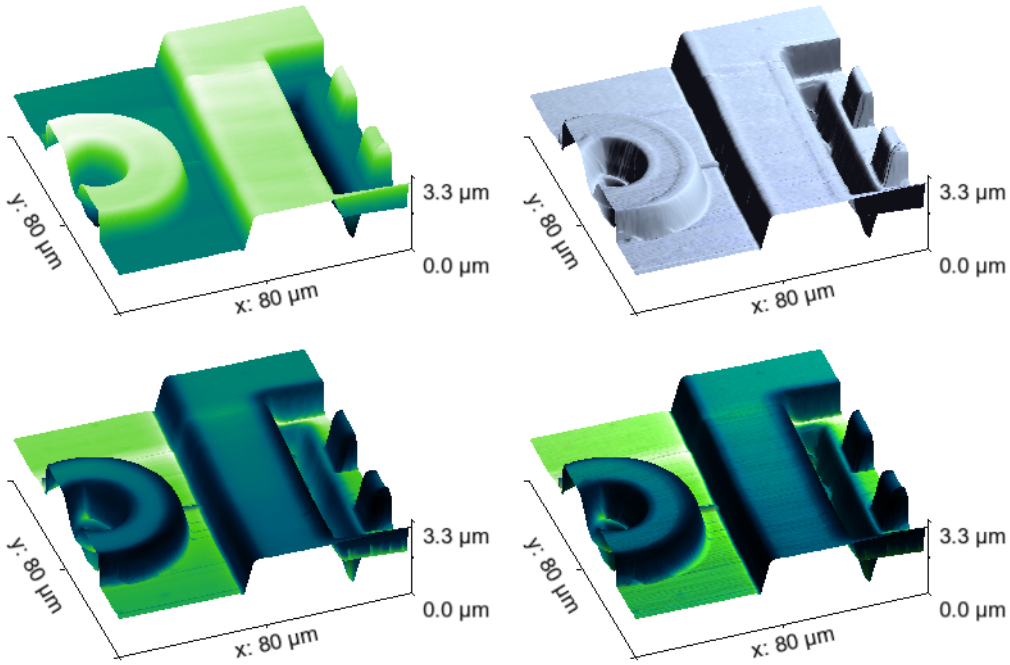
- *Basic* – controls to set rotations, scales and axis line width numerically and to switch on and off axes, axis labels, perspective projection and display of masked areas.

- *Light & Material* – visualization settings. This tab also contains controls to set light position numerically.
- *Labels* – fine tuning of sizes, positions, and other properties of axis labels.
- *Colourbar* – settings for the false colour bar which can be displayed on the right side of the 3D view.

Gwyddion 3D view has several visualization modes, selected in *Light & Material*, that combine two basic methods:

- False colour representation of data values, similar to normal image view.
- Rendering illumination of the surface with a light source (defined by the light controls).

Modes *Gradient* and *Overlay – no light* do not use illumination. The only difference between them is that in the simple gradient mode the colours always corresponds to the heights; in the overlay mode different images can be used for heights and colours. In *Lighting* mode the surface illumination is rendered using the selected material. Finally, *Overlay* combines colouring and lighting. Again, heights and colours can be given by different images.



Visualisation modes of the 3D view. Top row: simple gradient and lighting. Bottom row: overlays with a different channel used for colouring (left without lighting, right with lighting).

Saving Images

The 3D view can be saved into a bitmap image using the *Save* button. The image has exactly the same size and contents as displayed on the screen – except for possible removal of empty borders, if enabled with *Autocrop* in the *Basic* tab.

The default image format is PNG (Portable Network Graphics). By changing the file extension to `.jpg`, `.jpeg`, `.tif` or `.tiff` you can change the image format to JPEG or TIFF. Entering an unrecognised extension will save a PNG image, albeit with a confusing extension.

Note due to the peculiarities of certain operating systems, graphics drivers and windowing environments, artefacts may sometimes appear on the exported image in parts corresponding to obscured parts of the 3D view. If you encounter this problem, make sure the 3D view is not obscured by other windows during the image export.

OpenGL Material Editor

OpenGL material editor can be invoked with *Edit* → *GL Materials*. The controls in the material list are the same as in the colour gradient editor list and the material management works identically. The actual editor is of course different. It allows to edit four quantities defining the material:

- ambient colour $k_{\mathrm{a},\alpha}$ (where $\alpha = \mathrm{red}, \mathrm{green}, \mathrm{blue}$), controlling the reflection of ambient light that is assumed coming uniformly from all directions,
- diffuse colour $k_{\mathrm{d},\alpha}$, describing the diffuse reflection which is independent on the direction of incident light and whose apparent brightness is independent of the viewing angle,

- specular colour $k_{s,\alpha}$, controlling the specular reflection with reflected light intensity dependent on the angle between the observing direction and the direction of light that would be reflected by an ideal mirror with the same normal, and
- shininess s, a numeric exponent determining how much the specular reflection resembles an ideal mirror, smaller values mean rougher surfaces, higher values mean smoother surfaces.

If we denote **L** the normal vector pointing from the observed surface point to the light source, **V** the normal vector to the observer, **N** the normal vector to the surface and **R** the normal vector in the direction of ideal mirror reflection, the observed light intensity in OpenGL lighting model can be expressed as

$$I_\alpha = k_{a,\alpha} I_{a,\alpha} + k_{d,\alpha} I_{d,\alpha} (\mathbf{N} \cdot \mathbf{L}) + k_{s,\alpha} I_{s,\alpha} (\mathbf{R} \cdot \mathbf{V})^s$$

where $I_{a,\alpha}$, $I_{d,\alpha}$ and $I_{s,\alpha}$ are the ambient, diffuse and specular light source intensities, respectively.

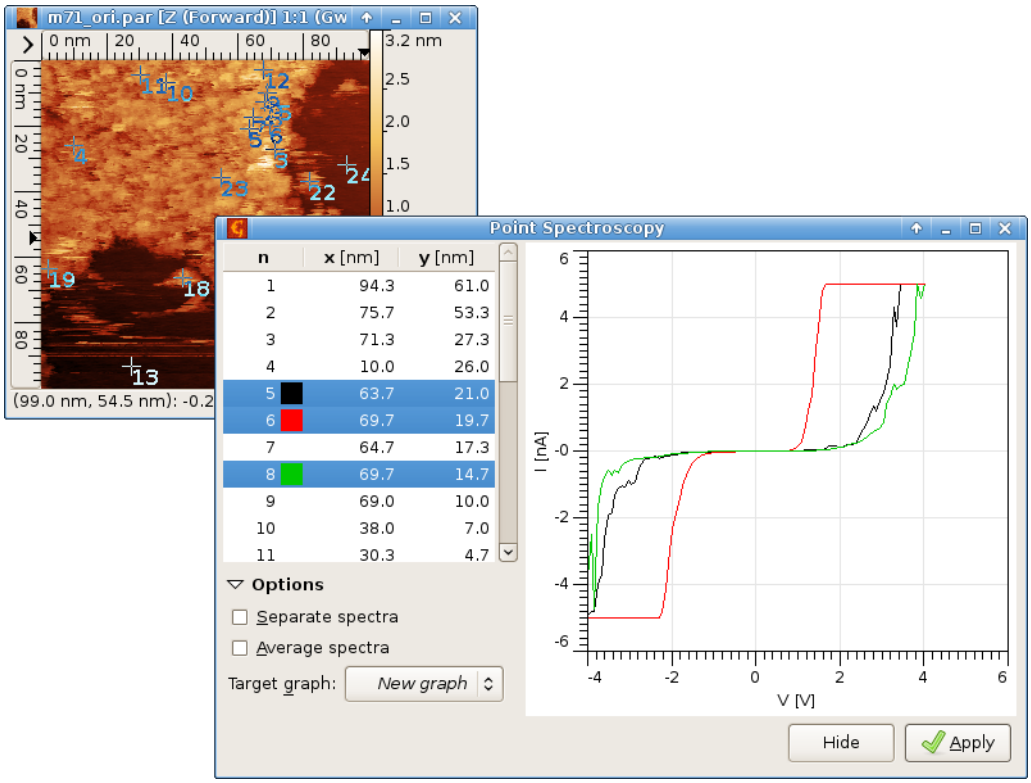
3.11 Single Point Spectra

Gwyddion currently offers some basic visualization and extraction means for single point spectroscopy data (we will generally refer to any curves measured in or otherwise attached to individual points of the sample as to "spectra" here). If spectra import is supported for a file type, they will appear in the *Spectra* tab of the data browser. Standalone spectra files can be added to the two-dimensional data using file merging.

Point Spectroscopy Tool

The primary spectra visualization and extraction tool is the Point Spectroscopy tool. It displays a list of measurement points and shows their positions on the data window. Individual curves can be selected either in the list or by selecting the corresponding crosses on the data window. If the file contains more than one spectrum set, one can choose among them by selecting the desired one in the *Spectra* tab list of the data browser.

The *Apply* button then extracts the selected set of curves into a graph that can be further analysed by graph functions, such as force-distance curve fitting.



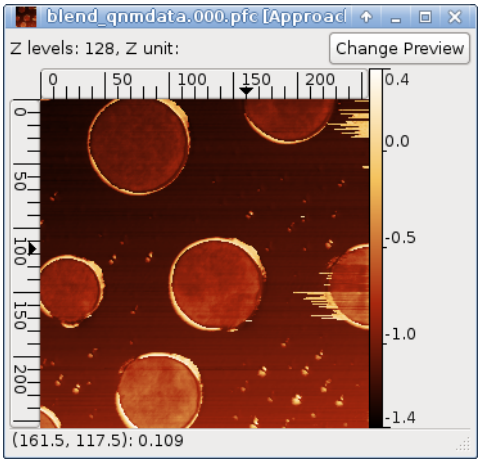
A data window with measurement points displayed and the point spectroscopy tool showing curves from three selected points.

3.12 Volume Data

Volume data are three-dimensional data representing, for instance, grid spectroscopy (spectra in each image point) or time evolution of the image. So, depending on the point of view, they can be imagined as either a stack of identically-sized images

or as a set of curves forming a regular grid (image). This differentiates volume data from curve maps in which each curve is different they cannot be viewed as a stack of images.

Gwyddion currently offers some basic visualization and extraction tools in the *Volume data* menu. Volume data windows, as shown in the following figure, provide the most basic visualisation of volume data: an "preview" image, representing one plane of the volume data or other associated two-dimensional data. The x and y coordinates of the image correspond to the x and y coordinates of the volume data; the z coordinates can be imagined as orthogonal to the screen.



A volume data window displaying the associated two-dimensional preview data.

Button *Change Preview* near the top of the window can be used to change the preview image. The preview can either be calculated as a summary quantity characterising data along the invisible z direction (minimum, maximum, mean, ...), chosen as a section of the volume data, alternatively just any two-dimensional data of the same dimensions can be shown as the preview.

3.13 Curve Maps

Curve maps are bundles of curves measured in points of a regular grid. While the grid is regular, like in image and volume data, curves in each bundle can have different numbers of points. Some bundles can even be empty (no data were measured in the corresponding point). Curve maps also usually comprise several data series, for instance height and force, whereas in other types of data each physical quantity is a separate data item.

Containing a bundle of curves in every point means that also the independent variable is one of the curves. For example, the force-distance curve bundle contains the force values and distance values. In most of the modules using curve maps one needs to choose which variable is dependent and which independent (sometimes the module can guess it).

An important concept is also the segmentation of the curve bundles. In some cases it is useful to have some marks on the bundle, e.g. to separate the approach and retract part of a force-distance curve. Such marks form segments which are then used by data processing modules. Segments are individual to curve bundles, so each bundle can use only one set of marks. Segments can be loaded from manufacturer's file format, or created in Gwyddion.

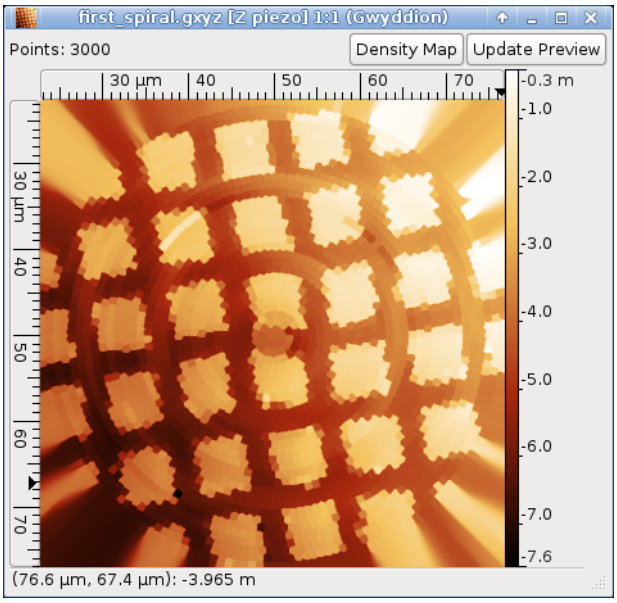
Button *Change Preview* near the top of the window can be used to change the preview image. The preview can either be calculated as a summary quantity for one of the curves characterising data along the invisible z direction (minimum, maximum, mean, ...), or alternatively just any two-dimensional data of the same dimensions can be shown as the preview.

3.14 XYZ Data

XYZ data are point clouds representing measurements in arbitrary sets of points in the xy plane. In SPM context it is generally assumed the data define a function in the plane, i.e. one point in the xy plane correspond to one z value, at least conceptually. More general XYZ data can represent arbitrary three-dimensional shapes. However, this is unusual in SPM and Gwyddion currently assumes the data correspond to a single-valued surface.

One important consequence is that XYZ data sets that vary in z values but are defined on the same points in the xy plane are considered "compatible" in Gwyddion. Many operations can be performed more directly with compatible data, or only make sense with compatible data.

The XYZ data processing options, found in the *XYZ Data* menu, are at present limited. XYZ data windows, as shown in the following figure, provide the basic visualisation of XYZ data: a preview raster image. The preview image is calculated using a fast method that provides a result close to nearest neighbour interpolation of the XYZ data. Resizing the window only resizes the preview image. Use the *Update Preview* button to recalculate the preview image for the current window size.



An XYZ data window displaying the preview image of a spiral-path scan.

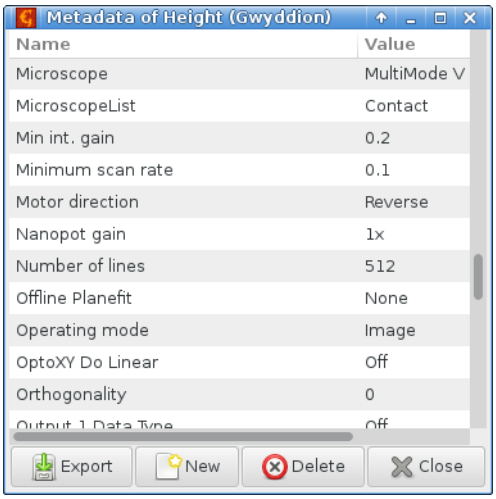
It is also possible to switch between preview of the data and display of the XYZ point density using the *Density Map* toggle button. The behaviour with respect to window resizing is the same as for the data.

3.15 Metadata

Auxiliary information and values describing certain data and the conditions it was measured on are called metadata in Gwyddion. They may include the SPM mode, tip bias, scanning frequency, measurement date and time or user comments.

Metadata are always per-channel. The metadata for the current image, volume or XYZ data can be displayed with *Metadata Browser* command in the right-click context menu (in version 2.32 or newer) or using *Meta* → *Metadata Browser* (in older versions). The browser lists all available metadata as Name, Value pairs. It also enables to modify and delete values or add new ones. It is possible to export all metadata of a channel to a text file with *Export* button.

The level of metadata support differs widely between file formats. For file formats that are well documented and/or allow to import all metainformation generically lots of auxiliary data including obscure hardware settings can be listed. On the other hand it is possible that no metadata is imported from your files, for example when they contain no auxiliary data or it is not known how to read it.



Metadata browser showing the metadata of a Nanoscope file.

3.16 Image Export

Microscopy data often need to be rendered into image formats, usually for presentation purposes, occasionally to open them in programs that do not support any SPM data format. Both is achieved the same way in Gwyddion: by selecting *File* → *Save As* and choosing a file name corresponding to an image format, e.g. `channel.png` or `channel.tiff`. The reference section High-Depth Image Formats describes the export of data as high-depth greyscale images that should be still usable as quantitative data. For supported formats you can enable it using the *Export as 16 bit greyscale* check-box at the top of the image export dialogue. The rest of this section discusses the creation of nice images for publications and presentations.

Gwyddion can render images into a number of formats, including for instance PNG, PDF, SVG, EPS, TIFF, WebP, BMP, PPM, TARGA and JPEG. Depending on the intent, some may be more suitable than others. Generally, the following choices can be recommended:

- PDF (Portable Document Format) for high-quality rendering suitable for printing, with all text and lines perfectly sharp and clean at any scale. Gwyddion also supports output to EPS (Encapsulated PostScript) for the same purpose, however, some features currently seem to work better in the PDF output.
- PNG (Portable Network Graphics) for web pages, low-resolution previews, thumbnails and icons, and also if you have to use a raster image format because it is the only option. PNG is a modern and widely supported raster image format with good loseless compression and a number of nifty features, including full transparency support.
- SVG (Scalable Vector Graphics) for subsequent editing and processing. SVG is a modern vector graphics format. You can open in a vector image editor such as Inkscape and modify it or combine with other images – keeping all text and lines perfectly sharp and clean.

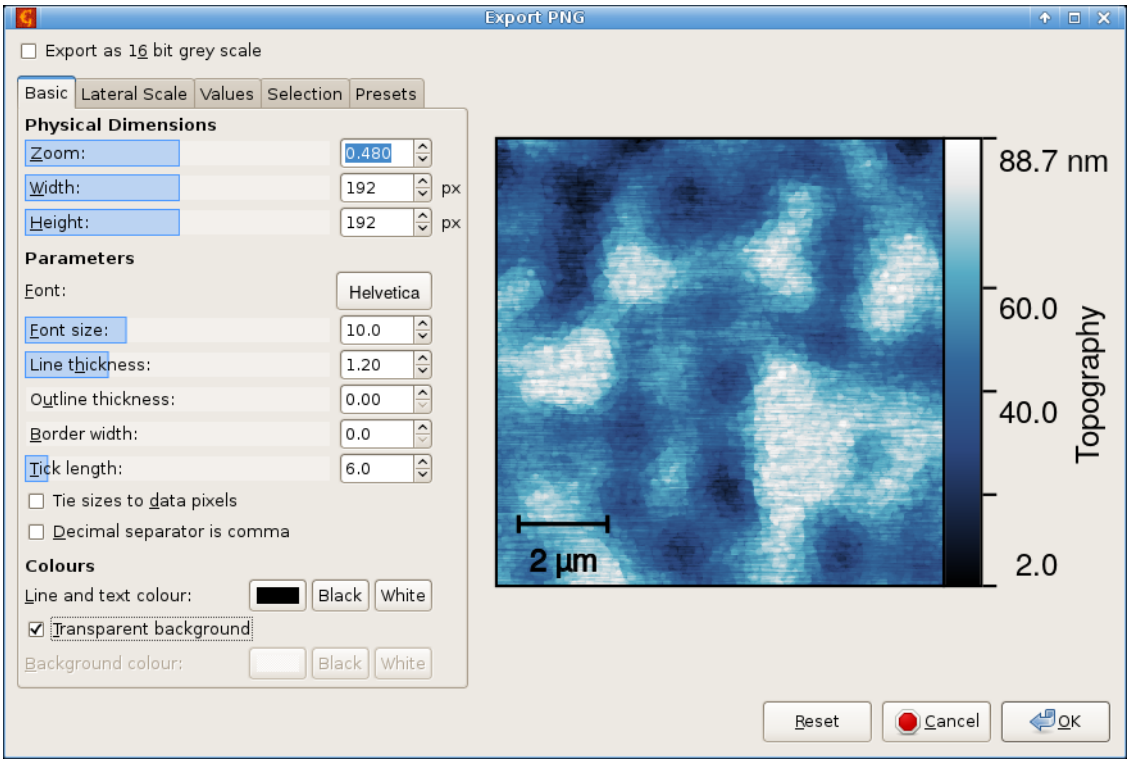


Image export dialogue screenshot showing the Basic *options page.*

The image export options are divided into several groups, as seen in the above screenshot.

Basic

Basic options specify various overall sizes and scales. The *Physical Dimensions* part differs for raster and vector images. For raster images the physical dimensions are specified as follows:

Zoom Scaling of data pixels to image pixels. The default zoom of 1 means data pixels correspond exactly to image pixels. Upscaling is possible with zoom larger than 1, and downscaling with zoom smaller than 1. For data with non-square pixels displayed with physical aspect ratio, the zoom is applied to the shorter side of the pixel, whichever it is.

Width Width of the rectangle corresponding to data in the exported image (not width of the entire image), in pixels.

Height Height of the rectangle corresponding to data in the exported image (not height of the entire image), in pixels.

Vector images do not have finite pixel resolution, therefore, the physical dimensions can be instead given as follows:

Pixel size Size of one data pixel in millimetres.

Pixels per inch Number of data pixels per one inch.

Width Width of the rectangle corresponding to data in the exported image (not width of the entire image), in millimetres.

Height Height of the rectangle corresponding to data in the exported image (not height of the entire image), in millimetres.

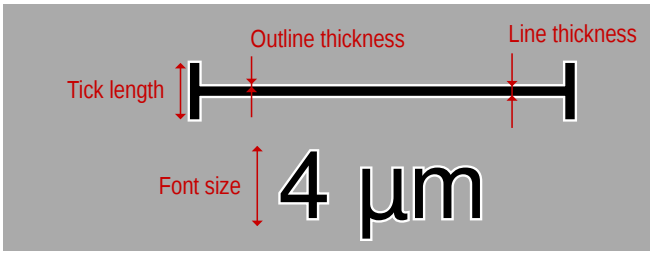


Illustration of the basic size parameters.

The remaining size and scale parameters, denoted *Parameters*, are common to both image types. However, for raster images the values are in pixels while for vector images they are in typographical points. More precisely, this is true if they are given as absolute, i.e. the option *Tie sizes to data pixels* is not selected. If this option is selected all sizes are relative to data pixels, i.e. they scale together with the image data when the physical dimensions change. The parameters, illustrated in the figure above, include:

Font The font used to draw all labels.

Font size Font size, in pixels or typographical points as described above.

Line thickness Thickness of lines: borders, ticks, inset scale bar and selections.

Outline thickness Thickness of outlines that can be drawn around the inset scale bar and selections.

Border width Width of empty border around the entire image.

Tick length Length of ruler and false colour map ticks. This setting also controls the inset scale bar ticks and crosses drawn for the point-like selection.

Option *Decimal separator is comma* permits controlling the format of numbers. By default, decimal dots are used even if your locale setup specifies commas. Enabling this options replaces the dots with commas, again independently on the actual locale.

Finally, *Colour* options permit controlling common colours. *Line and text colour* controls the colour of all lines and labels drawn outside the image area, that is over the background. The background can be transparent (enabled with *Transparent background* check-box) if the target image format supports it – this currently includes PNG, WebP and all vector formats. Otherwise the background is solid colour can be changed using the *Background colour* control.

Lateral Scale

Settings in the *Lateral Scale* page control how the lateral dimensions are visualised. There are two basic choices displayed in the figure below, rulers and an inset scale bar. The lateral scale can be also disabled entirely. The inset scale bar has the following settings:

Length The bar length can be set manually to an arbitrary value which does not result in a too short or too long bar. Press **Enter** to update the preview when you modify the length. Button *Auto* selects a suitable length automaticaly (this is also done when the manually entered length is not found reasonable).

Placement The bar can be placed along the upper or lower edge and aligned to either side or centered.

Horizontal gap Horizontal gap between the bar and the closer vertical edge of the data area (meaningful only if the bar is not centered).

Vertical gap Horizontal gap between the bar and the closer horizontal edge of the data area.

Colour Colour with which the bar is drawn. Buttons *Black* and *White* allow choosing quickly the basic colours.

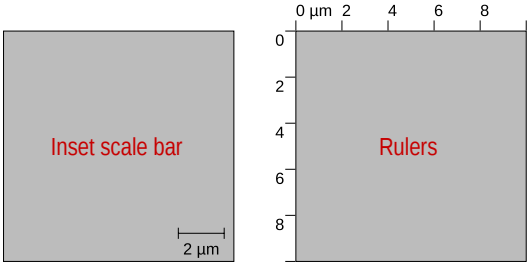
Outline colour Colour of the outlines. Note if the outline width is zero (the default), no outlines are drawn, hence changing the colour has no effect.

Opacity Opacity with which the inset scale bar is drawn. This setting permits drawing the bar as semi-transparent.

Draw ticks If enabled, the bar has vertical ticks at its ends. When disabled, the bar is just a line.

Draw label If enabled, the bar length is displayed under or above the bar.

Draw text above scalebar If enabled, the bar length is displayed above scalebar, otherwise below.



Lateral scale visualisation types.

Value

Settings in the *Value* page control the rendering of values and false colour mapping. Two basic settings control the rendering of field data:

Interpolation The interpolation type can be noticeable particularly for large zooms. In the case of vector image formats, the final rendering is done when then the image is viewed or printed. Hence the available interpolations are limited to two types. Round, in which each data pixel is drawn as a sharp rectangle, and Linear, in which values in the rendered image are linearly interpolated between original data pixels. In the case of raster images, you can choose from the full set of interpolations supported by Gwyddion because the interpolation is done during the export.

Draw mask If enabled, the mask is drawn on top the data using the same colour as in the data window.

When a mask is present and shown then optionally a mask legend can be added below the image as illustrated in the following figure if *Draw mask legend* is enabled. The label can be specified arbitrarily using the *Label* entry and the spacing is controlled by *Vertical gap*.

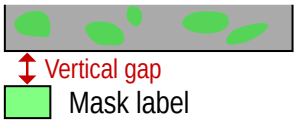
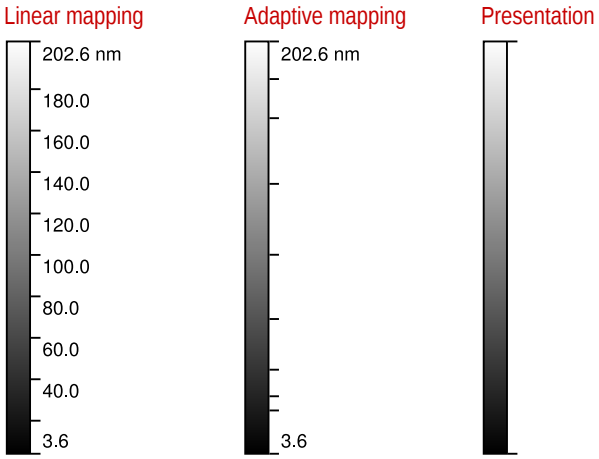


Illustration of the mask label placed below the image and parameters controlling it.

The value scale can be rendered as a false colour ruler or disabled. The ruler is drawn somewhat differently depending on the false colour mapping type, as illustrated in the following figure.



Rendering of the false colour map scale depending on the mapping type. For the standard linear mapping, whether full or fixed range, ticks with values are drawn. For an adaptive mapping the colour gradient is drawn the same, but tick positions correspond to the adaptive mapping and interior ticks are drawn without labels (to avoid overlapping labels). If a presentation is shown, the values are considered arbitrarily scaled thus no values and interior ticks are displayed.

It is possible to set the number of decimal places for the tick marks by enabling *Fixed precision* and specifying the number. Otherwise a suitable precision is determined automatically.

The automatic choice of units for the false colour map scale can be adjusted using *Fixed kilo threshold*. It gives the threshold for switching to the next power of 10^3, e.g. from nm to μm or from pA to nA. For instance, setting it to 400 means that if the maximum absolute height is 399 nm it will be displayed as 399 nm, whereas if it is 401 nm it will be displayed as 0.401 μm. The threshold applies to all powers of 10^3 so for the same setting the units would switch from mV to V at 400 mV or from pA to nA at 400 pA.

The image title can be optionally added to the top of the image or along the false colour ruler. Using the check-box *Put units to title*, the placement of value units can be chosen between the title and false colour ruler. Note if the no title is drawn then placing units to the title disables them entirely.

Settings *Horizontal gap* for the false colour ruler and *Gap* for the title control the gaps between them and the corresponding image edges as shown in the following figure. If the title is drawn along the false colour ruler the gap can be negative, moving the title a bit inside the ruler. For titles along the top edge, negative gap values are ignored.

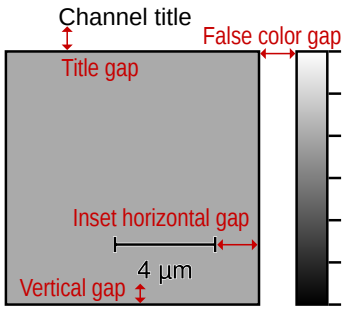


Illustration of the various gap settings (for image title, in the position at the top).

Selection

Any kind of selection stored with the data can be also shown in the image. If the check-box *Draw selection* is enabled you can choose the selection to draw from the list below. The colours are specified in the same manner as for the inset scale bar:

Colour Colour with which the selection is drawn. Buttons *Black* and *White* allow choosing quickly the basic colours.

Outline colour Colour of the outlines. Note if the outline width is zero (the default), no outlines are drawn, hence changing the colour has no effect.

Opacity Opacity with which the selection is drawn. This setting permits drawing the selection as semi-transparent.

Beside the colours, some selection types have further options, for instance whether individual shapes are numbered. If you are using a tool that has some shapes selected on the data, the kind of the selection to draw and the specific options are preset to match the current tool. This is usually the most convenient way to get the selection drawn as you intend. However, any existing selection can be drawn and the options adjusted manually if you wish.

Presets

Different sets of image rendering options are useful on different occasions. A set of options can be saved as a preset and recalled later. The list in the *Presets* page shows all saved presets that can be managed using the buttons below:

Load Loads the currently selected preset, i.e. sets the image rendering options according to the preset. The inset scale bar length can be set to the automatic value if the stored length is not found reasonable. Also the selection type and its options are kept intact, only the colours are set according to the preset.

Store Stores the current options under the name given as *Preset name*. The name also serves as a file name so it is advisable to avoid odd characters in the preset name. If a preset of the same name already exists, it is overwritten.

Rename Renames the currently selected preset to the name given as *Preset name*. The preset does not have to be loaded for renaming. It is not possible to overwrite an existing preset by renaming.

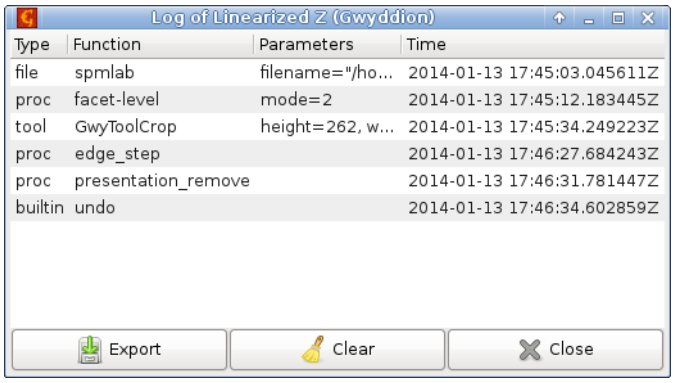
Delete Deletes the currently selected preset.

3.17 Logging

Gwyddion records data modification operations for each image, volume or XYZ data in so-called log. When the data are saved to a `.gwy` file, the log is saved along with them. The log can be displayed by selecting the *View Log* command in the right-click context menu of the corresponding data window. This is useful for recalling later what corrections you applied, how a mask or presentation was created, etc. It should be noted that the logs are informational; they are neither intended nor suitable for auditing purposes.

The log view is live: if you keep it open you can see individual data processing operations appearing there as you perform them.

A simple log example is shown in the following figure. For each operation, the type, name, parameters and time are recorded. The type can be for instance file import, data processing function or tool application. The function names correspond to those shown in the module browser (*Info* → *Module Browser*), where they are listed in *Registered functions* for each module; or in the on-line module browser. The parameter list represents the settings of the function at the time it was used. Since the log is only informative, the parameters may or may not allow a precise reconstruction of the operation. For typical simple operations, they should be sufficient. In the case of complex or interactive operations involving multiple data, the log entry may not be detailed enough. The time is recorded in the local time zone when the operation is performed; it is not recalculated when you send the files around the world and display elsewhere.



Log viewer showing a simple data processing operation log for an image, starting with file import from an SPM vendor format and continuing with the application of data processing functions, tools and undo.

The entire log can be exported to a text file using the *Export* button. It not possible to modify the log entries as that would defeat the purpose of logging somehow, but you can clear the entire log with *Clear*.

Disabling logging

In some circumstances, you may wish to store or publish `.gwy` files without logs. Therefore, logging is controllable on several levels:

- It can be enabled and disabled globally using *Edit* → *Logging Enabled*. When logging is disabled no new log entries are added. Existing logs are not removed though and you can still view them. They are also still saved to `.gwy` files.
- The log for a specific image, volume or XYZ data can be cleared with button *Clear* in the viewer.
- All logs in the current file can be removed using *File* → *Remove All Logs*. As with any other file modification, the file needs to be saved afterwards for the log removal to have any effect on the on-disk file. And, of course, if logging is enabled and you start modifying the data, new logs will be created and the new data operations recorded in them.

3.18 Raw Data File Import

Both raw ASCII and binary data files and files in unsupported formats can be imported with rawfile module – with some effort. Raw data import can be explicitly invoked by selecting *Raw data files* type in the file open dialogue. It can be also set to appear automatically when you try to open a file in an unknown format. This is controlled in the raw file dialogue by option *Automatically offer raw data import of unknown files*.

Information

Its first tab, *Information*, allows to set basic file information:

Horizontal size*, *Vertical size Horizontal and vertical data resolution (number of samples).

Square sample Fixes horizontal and vertical resolution to the same value.

Width*, *Height Physical sample dimensions.

Identical measure Keeps the ratio between physical dimension and number of samples equal for horizontal and vertical direction, that is the data has square pixels.

Z-scale (per sample unit) The factor to multiply raw data with to get physical values.

Missing value substitute Special value that denotes missing data. Often a value at the edge of raw data range is used, e.g. −32768 for signed 16bit data. For text data it is also possible to give a string, e.g. `BAD` or `NODATA`, that substitutes missing data in the file.

If the missing value handling is enabled all pixels with the specified substitute value will be masked and replaced with a neutral value in the imported data field.

Invalid values (not-a-number, NaN) in floating point data types are masked and replaced automatically so you only need to enable the replacement if some other (finite) value is used to denote missing data.

Data Format

On the second tab, *Data Format*, particular data format can be chosen. There are two independent possibilities: *Text data* and *Binary data*.

Text files are assumed to be organized by lines, each line containing a one data row, data being represented by integers or floating point numbers in standard notation. Following options are available:

Start from line The line data starts at, that is the number of lines to ignore from file start. All types of end-of-line markers (Unix, MS-DOS, Macintosh) are recognized.

Each row skip The number of fields to ignore at the beginning of each line.

Field delimiter, Other delimiter If delimiter is *Any whitespace*, then any non-zero number of whitespace characters counts as field delimiter. If a whitespace character is selected, the delimiter must be this character. Otherwise field are separated by specified character or string and all whitespace around delimiters is ignored.

Decimal separator is comma By default, floating point numbers are assumed to use decimal point. This option changes it to comma.

Following options are available for binary files:

Binary data You can either select one of predefined standard data formats, or *User defined* to specify a format with odd number of bits per sample or other peculiarities.

Byte swap pattern How bytes in samples are swapped. This option is only available for predefined formats larger than one byte. Its bits correspond to groups of bytes to swap: if the j-th bit is set, adjacent groups of 2^j bits are swapped.

For example, value 3 means sample will be divided into couples (bit 1) of bytes and adjacent couples of bytes swapped, and then divided into single bytes (bit 0) and adjacent bytes swapped. The net effect is reversal of byte order in groups of four bytes. More generally, if you want to reverse byte order in groups of size 2^j, which is the common case, use byte swap pattern $j-1$.

Start at offset Offset in file, in bytes, the data starts at.

Sample size Size of one sample in bits for user defined formats. E.g., if you have a file with only 4 bits per sample, type 4 here. For predefined formats, their sample size is displayed, but it is not modifiable.

After each sample skip The number of bits to skip after each sample.

Usually, samples are adjacent to each other in the file. But sometimes there are unused bits or bytes between them, that can be specified with this option. Note for predefined types the value must be a multiple of 8 (i.e., only whole bytes can be skipped).

After each row skip The number of bits to skip after each sample in addition to bits skipped after each sample.

Usually, rows are adjacent to each other in the file. But sometimes there are unused bits or bytes between them, that can be specified with this option. Note for predefined types the value must be a multiple of 8 (i.e., only whole bytes can be skipped).

Reverse bits in bytes Whether the order of bits in each byte should be reversed.

Reverse bits in samples Whether the order bits in each sample should be reversed for user defined samples.

Samples are signed Whether samples are to be interpreted as signed numbers (as opposed to unsigned). For predefined formats, their signedness is displayed, but it is not modifiable.

Presets

Import settings can be saved as presets that allow to easily import the same file – or the same file type – later.

Button *Store* saves current import settings under the name in *Preset name* field. *Rename* renames currently selected preset to specified name, *Delete* deletes selected preset, and *Load* replaced current import setting with selected preset.

3.19 Specific Data Import

Import of several other types of data is not automatic and it requires human intervention.

Graphics Formats

Importing data from image formats such as PNG, TIFF, JPEG or BMP is similar to import from raw/unknown file formats, only simpler.

It is simpler because the file structure is known and the file format is automatically detected. Hence the file type does need to be selected explicitly. However, the data interpretation is still unknown and must be specified manually. The pixmap import dialogue therefore resembles the *Information* tab of raw data import, requiring you to set the physical dimensions and value scale.

Note the physical dimensions suggested there are *not* obtained from the file, they are simply the last values used. Some SPM data formats are based on an image format (typically, TIFF is used as the base) and contain the information about physical scales and units, albeit stored in a manufacturer-specific way. In this case a separate import module can be written for this particular format to load the files automatically with correctly scaled values.

See the reference section High-Depth Image Formats for the details of support for high-depth images and the possibility of using them for data representations.

For plain (non-SPM) images there is also an option to *Just use pixels* for the dimensions. This sets the physical dimensions equal to the pixel dimensions. They can be still adjusted later, for instance using Dimensions and Units.

Image stack as volume data

A multipage image (image stack) can be also imported as volume data instead of a heap of images. It is enabled by *Import as volume data*. This option currently exists only for multipage TIFFs.

The z coordinate in the imported data is simply the image number in the stack (starting from 0). Use volume data Dimensions and Units or Z Calibration function to set the physical scale and/or units of z.

Graph Curves

Simple two-column text files containing curve data can be imported as graph curves. In some cases, these files are recognized automatically. They can also be explicitly selected as *ASCII graph curve files* in the file open dialogue, causing the import module to try harder to load the file as a graph data.

The import dialogue shows a preview of the graph and permits to set the units and labels.

XYZ Data

Three-column text files containing XYZ data are imported by selecting the *XYZ data files* file type. Again, they can be recognized automatically but requesting this format explicitly makes the module to try harder to load the file as XYZ data.

The import dialogue displays the number of points and physical ranges of the data and permits setting the lateral and value units. The displayed ranges in physical units are updated according to the units you enter so you can immediately see if they they are correct.

Occasionally one can encounter image data are stored as XYZ data with completely regular lateral coordinates. Since they are still meant to be treated as image data, just in an odd format, the import module attempts to detect this situation. It then offers direct rendering to an image instead of importing the file as true XYZ data.

Nano Measuring Machine XYZ Data

The Nano Measuring Machine data are sets of often huge XYZ files that would be difficult, if even possible to handle normally in Gwyddion. Therefore, they are directly rasterised to an image upon import. Select the main description file (`.dsc`) to open the data file set and set the import parameters.

Usually, the vertical size should be chosen as the number of profiles (i.e. number of data files). However, a larger resolution can be also useful. If you enable the *Identical measures* checkbox then the imported image will have square pixels. Only one resolution can be then entered – the other one is determined automatically.

On 64bit systems, the import module can currently handle up to 2^{32} points per channel while the total number of data values is not limited. Note that reading the data points can take considerable time. Select only channels you are actually interested in to speed up the import process.

Nanonis SXM Data

Nanonis uses a right-handed coordinate system and Gwyddion a left-handed system. Hence it is not possible to preserve simultaneously how images look visually and the absolute coordinates of images and their features – the latter is required for correct import of spectra.

The default behaviour was varying between versions. Since version 2.52 you can choose it by setting the value of `/module/nanonis/pres` in the settings:

- `False` means images look the same, but vertical coordinates are flipped.
- `True` means coordinates are correct, but images look flipped vertically.

3.20 Text Data Export

Exporting data to plain text file can be often useful for further processing in other software as text files are highly portable.

Image data Image data can be exported simply by invoking *File* → *Save as* and selecting file name extension `.txt` or choosing *ASCII data matrix (.txt)* from the file save dialogue. This writes the current channel as a matrix of values, each line in the file corresponding to one image row. Optionally, the export of all channels can be concatenated and written to one file (separated by blank lines).

Exactly the same method allows also exporting image data to simple three-column text files as XYZ data. Just select file name extension `.xyz` or choose the *XYZ text data (.xyz)* format instead.

Graph curves Graph curves are exported using Export Text from either the *Graph* menu or graph right-click context menu.

Volume data Similarly, volume data are exported using Export Text from the *Volume Data* menu.

Tabular Data

A number of functions which calculate lists and sets of values offer their export as tabular data. The following figure shows an example from the Statistical quantities tool.

Controls for tabular data export, from left to right: scientific number format switch, data format selector, copy to clipboard and save to file buttons.

The scientific number format switch controls whether numbers are written in the machine-readable scientific format, with decimal dot (regardless of locale settings) and `e` for power of 10. When it is switched on, the values are also written in base units – metres, as opposed to micrometres or kilometres, for instance. This means output like

```
Minimum: 3.32e-9 m
Maximum: 1.20e-9 m
```

With format intended to be human-readable (scientific format switched off) the same values could be formatted

```
Minimum: 3,32 nm
Maximum: 1,20 nm
```

if Gwyddion runs in an environment where comma is used as the decimal separator.

The overall format of tabular data is controlled by the data format selector with the following choices (not all may be present if they do not make sense in particular context):

Colon: Somewhat free format intended mainly to be human-readable. The label is separated by a colon and the value and unit are simply printed together and aligned:

```
Maximum peak height (Sp): 6.6754 nm
Maximum pit depth (Sp):   6.1039 nm
Maximum height (Sz):      12.7793 nm
```

TAB Fixed number of columns – typically name, value, error, unit – separated by the TAB character (ASCII 9). The previous example could be formatted:

```
Maximum peak height (Sp)↦6.6754↦↦nm
Maximum pit depth (Sp)↦6.1039↦↦nm
Maximum height (Sz)↦12.7793↦↦nm
```

where ↦ stands for the TAB character. Note the empty columns for value errors. If the table can contain errors under some circumstances, which is for statistical quantities the case when measurement uncertainties are present, the corresponding columns appears always, even if they are empty. This prevents format inconsistency with and without errors.

CSV Comma-separated values, a common (although not really standardised) format with a fixed number of columns separated by commas. Sometimes semicolons are also used despite the name, however, Gwyddion writes files with comma as the separator. The previous example could be formatted:

```
"Maximum peak height (Sp)","6.6754","","nm"
"Maximum pit depth (Sp)","6.1039","","nm"
"Maximum height (Sz)","12.7793","","nm"
```

Chapter 4

Data Processing and Analysis

The number and breadth of data manipulation and analysis functions is one of the Gwyddion main strengths. The description of their principles and applications is the purpose of this chapter, with focus on the explanation of how they work and what they calculate or perform. Elements of their user interface are also occasionally described where it seems useful, nevertheless, it is assumed the reader is familiar with the basic organization of data and controls in Gwyddion, as described in the previous chapter.

4.1 Basic Operations

Value-reading and basic geometrical operations represent the core of any data processing program. Gwyddion offers a wide set of functions for data scaling, rotation, resampling or profile extraction. This section describes these simple but essential functions.

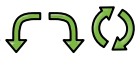
Most basic operations can be found in *Data Process* → *Basic Operations* menu; some are also carried out by tools.

Elementary Geometrical Transformations

Geometrical transformations in this group preserve pixels, i.e. a pixel in the transformed image corresponds precisely to one pixel in the source image. There is no interpolation.

Flip 

Flip the data horizontally (i.e. about the vertical axis), vertically (i.e. about the horizontal axis) or around the diagonal (i.e. exchanging rows and columns) with *Data Process* → *Basic Operations* → *Flip Horizontally*, *Flip Vertically* or *Flip Diagonally*, respectively.

Rotation by multiples of 90° 

Rotation of data by multiples of 90 degrees is done using some of the basic rotation functions: *Data Process* → *Basic Operations* → *Rotate Clockwise*, *Rotate Anticlockwise* or *Flip Both*. Flipping both axes rotates the image by 180°.

Crop 

The Crop tool can crop an image either in place or putting the result to a new image (with option *Create new image*). With *Keep lateral offsets* option enabled, the top left corner coordinates of the resulting image correspond to the top left corner of the selection, otherwise the top left corner coordinates are set to $(0, 0)$.

Extend 

Data Process → *Basic Operations* → *Extend*

Extending is essentially the opposite of cropping. Of course, adding more real data around the image borders is only possible by measuring more data. So this function offers, instead, several simple artificial extension methods such as mirrored and unmirrored periodic continuation or repetition of boundary values. The Laplace method works in the same way as Interpolate Data Under Mask and creates a smooth extension which converges to the mean value of the entire image boundary.

Transformations with Interpolation

Transformations in this group do not map one pixel to one pixel and require interpolation.

Scale 

Data Process → *Basic Operations* → *Scale*

Resample the data to different pixel dimensions (number of image columns and rows) using the selected interpolation method. The physical dimensions are unchanged.

If you want to specify the physical dimensions of one pixel, use Resample instead.

Square Samples 

Data Process → *Basic Operations* → *Square Samples*

Upsample the data (along the axis with larger pixel size) to make pixels square. Most scans have pixels with 1:1 aspect ratio, therefore this function has no effect on them.

Resample

Data Process → *Basic Operations* → *Resample*

Resample the data to different physical pixel size using the selected interpolation method. The pixel size can be selected to match another image, which is required for some multi-data operations. The physical dimensions are unchanged.

If you want to specify the number of image rows and columns or scaling ratio, use Scale instead.

Arbitrary rotation 

Function *Data Process* → *Basic Operations* → *Rotate* rotates the image by arbitrary angle. Unlike the simple geometrical transforms above, this requires interpolation, which can be specified in the dialogue.

After rotation, the image is no longer a rectangle with sides parallel to the axes. Option *Result size* specifies the area to take as the resulting image:

- *Same as original* preserves the image size, cutting off some data and adding some empty exterior data in the corners.
- *Expanded to complete data* ensures no parts of the image are cut off – which often means the results includes lots of empty exterior.
- *Cut to valid data* ensures the results contains no empty exterior – which often means cutting it considerably.

A mask can be optionally added over pixels corresponding to the exterior of the original image.

Binning 

Data Process → *Basic Operations* → *Binning*

Binning is an alternative method of size reduction. The resulting image is not created by interpolating the original at given points but instead by averaging all original pixels contributing to the result pixel.

The rectangular block of pixels that are averaged is the bin. You can control its dimensions using *Width* and *Height* that are also downscaling factors in each direction. The origin of the first bin can be offset from the image origin. Note, however, that the result is formed only from complete bins; incomplete bins at image edges are ignored.

Trimmed mean can be used instead of the standard average, with *Trim lowest* and *Trim highest* controlling how many lowest and highest values are discarded before averaging. This can be useful for suppressing local outliers. When *Sum instead of averaging* is enabled, the resulting value is calculated as the sum of all the contributions. This would make little sense for typical topographical data. However, if the image represents a density-like quantity whose integral should be preserved, summing is the correct operation.

Value Transformations

Functions in this group preserve pixels 1:1. However, they modify their values. See also the Arithmetic module which allows data manipulation using arithmetic expressions.

Invert Value 

Data Process → *Basic Operations* → *Invert Value*

The inversion function inverts the data about the mean value, keeping the mean value unchanged.

Limit Range 

Data Process → *Basic Operations* → *Limit Range*

Data range can be limited by cutting values outside the specified range. The range can be set numerically or taken from the false colour map range previously set using the Colour range tool and it is also possible to cut off outliers farther than a chosen multiple of RMS from the mean value.

Wrap Value 

Data Process → *Basic Operations* → *Wrap Value*

Periodic data, such as angles, are sometimes stored with an inconvenient wrap-around which makes them look discontinuous. For instance, the value range is from 0 to 360 degrees, but the data would be continuous if represented in range −180 to 180 degrees. Simple rewrapping then restores continuity.

The function has two parameters. *Range* is the value range (period in the z direction). *Offset* specifies the split value for rewrapping. The exact formula is

$$z_i \to \mathrm{fmod}(z_i - \Delta, r) + \Delta$$

Symbols r and Δ represent the range and offset, respectively. Function fmod is the floating point modulo (remainder) function.

Dimensions and Units 

Data Process → *Basic operations* → *Dimensions and Units*

Change physical dimensions, units or value scales and also lateral offsets. This is useful for the correction of raw data that have been imported with wrong physical scales or as a simple manual recalibration of dimensions and values.

The recalibrations can be specified either as absolute, i.e. setting the dimensions or value range to given values, or as corrections applying a factor and/or offset. In addition a few special modes exist which can for instance match exactly the pixel size to another image.

Since settings are remembered between invocations, multiple images can be quickly recalibrated the same way. For new, unrelated recalibrations make sure the properties you do not want to change are set to *Do not change* or use the *Reset* button.

Tilt 

Data Process → *Basic Operations* → *Tilt*

Tilt the data by a gradient or angle, specified numerically.

Reading Values

The simplest value reading method is to place the mouse cursor over the point you want to read value of. The coordinates and/or value is then displayed in the data window or graph window status bar.

Read Value Tool 

The Read Value tool offers more value reading possibilities: it displays coordinates and values of the last point of the data window the mouse button was pressed. It also displays a zoomed view of the image around this point.

The tool can average the values from a circular neighbourhood around the selected point, this is controlled by option *Averaging radius*. When the radius is 1, the value of a single pixel is displayed (as the simplest method does). Button *Set Zero* shifts the surface to make the current z the new zero level.

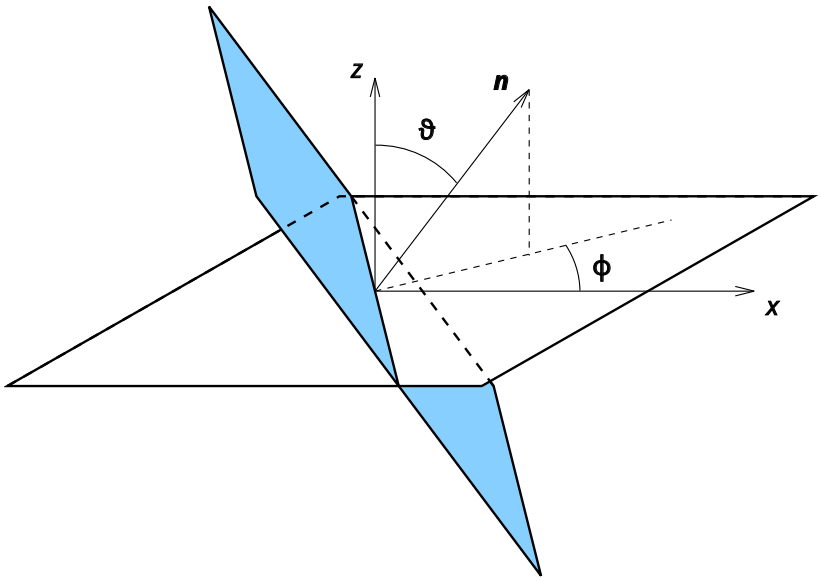
Read Value can also display the inclination of the local facet or local surface curvature. *Averaging radius* again determines the radius of the area to use for the plane fit. Curvature in particular need relatively large areas to be reliable.

Inclinations

In all Gwyddion tools, facet and plane inclinations are displayed as the spherical angles (ϑ, φ) of the plane normal vector.

Angle ϑ is the angle between the upward direction and the normal, this means that $\vartheta = 0$ for horizontal facets and it increases with the slope. It is always positive.

Angle φ is the counter-clockwise measured angle between axis x and the projection of the normal to the xy plane, as displayed on the following figure. For facets it means φ corresponds to the downward direction of the facet.

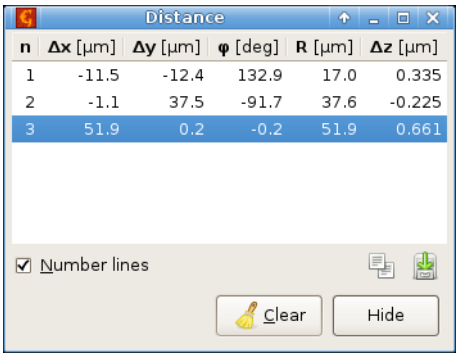


Surface facet (displayed blue) orientation measured as the anticlockwise angle from x-axis to the projection of facet normal vector **n** *to xy plane.*

Distance Tool 

Distances and differences can be measured with the Distance tool. It displays the horizontal (Δx), vertical (Δy) and total planar (R) distances; the azimuth φ (measured identically to inclination φ) and the endpoint value difference Δz for a set of lines selected on the data.

The distances can be copied to the clipboard or saved to a text file using the buttons below the list.



Distance tool with three selected lines.

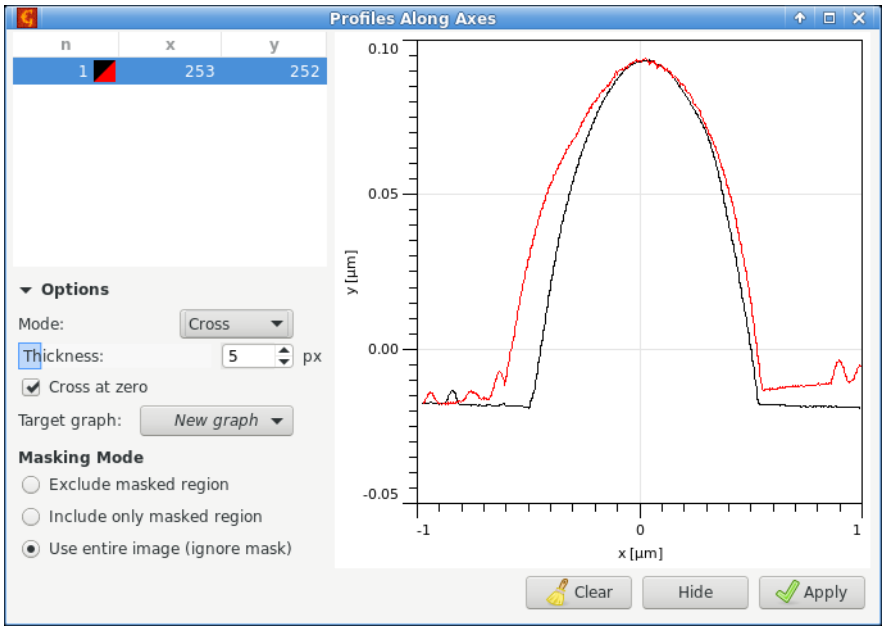
Profiles along axes 

The Profiles along axes tool extracts scan lines in horizontal and/or vertical direction. The direction is controlled by *Mode*, which can be *Horizontal*, *Vertical* or *Cross*. In the cross mode both profiles are extracted simultaneously.

When the *Cross at zero* option is enabled, the profile abscissa origin is in the crossing point (or tick mark for horizontal and vertical profiles). This can be useful for alignment of profiles in the graph. When the option is disabled, the profile coordinates correspond to image coordinates in the corresponding direction.

Profiles can be of different "thickness" which means that multiple scan lines around the selected one are averaged. This can be useful for noise suppression while measuring regular objects. Profile thickness is denoted by line endpoint markers in the image.

Scan lines from multiple images can be simulatenously read using Multiprofile.

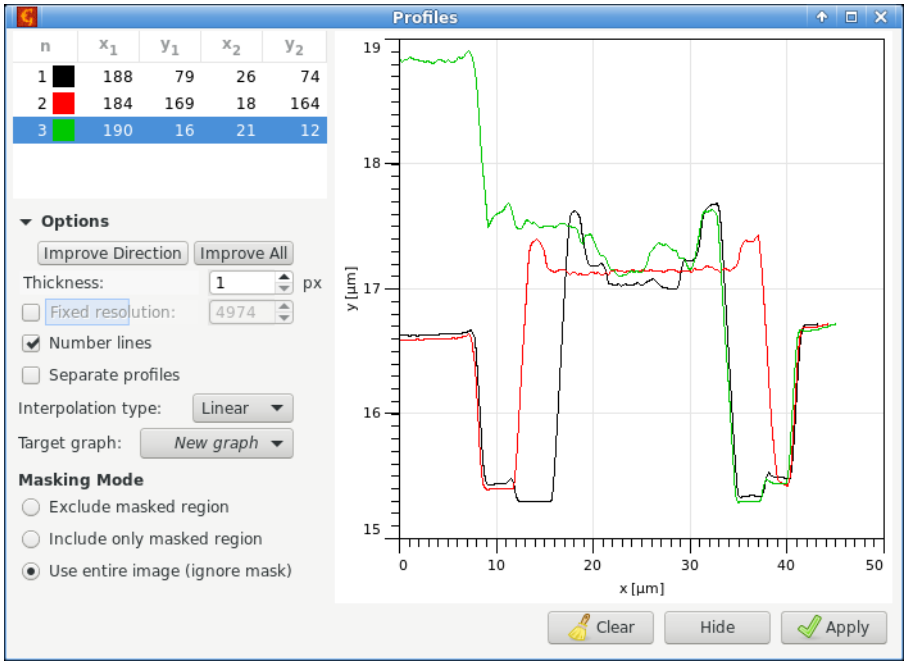


Profiles along axes tool with horizontal and vertical profiles crossing at zero and expanded options.

Profile extraction

The Profile tool extracts profiles along arbitrary lines drawn and adjusted using mouse in the image, shown in a live preview in the dialogue. Profiles can be of different "thickness" which means that more neighbour data perpendicular to profile direction are used for evaluation of one profile point for thicker profiles. This can be very useful for noise suppression while measuring regular objects. Profile thickness is denoted by line endpoint markers in the image.

After profiles are chosen, they can be extracted to graphs (separate or grouped in one Graph window) for further analysis using Graph functions.



Profile tool with three extracted profiles and expanded options.

The profile curve is constructed from data sampled at regular intervals along the selected line. Values in points that do not lie exactly at pixel centres (which normally occurs for oblique lines) are interpolated using the chosen interpolation mode. Unless an explicit number of samples to take is set using the *Fix res.* option, the number of samples corresponds to the line length in pixels. This means that for purely horizontal or purely vertical lines no interpolation occurs.

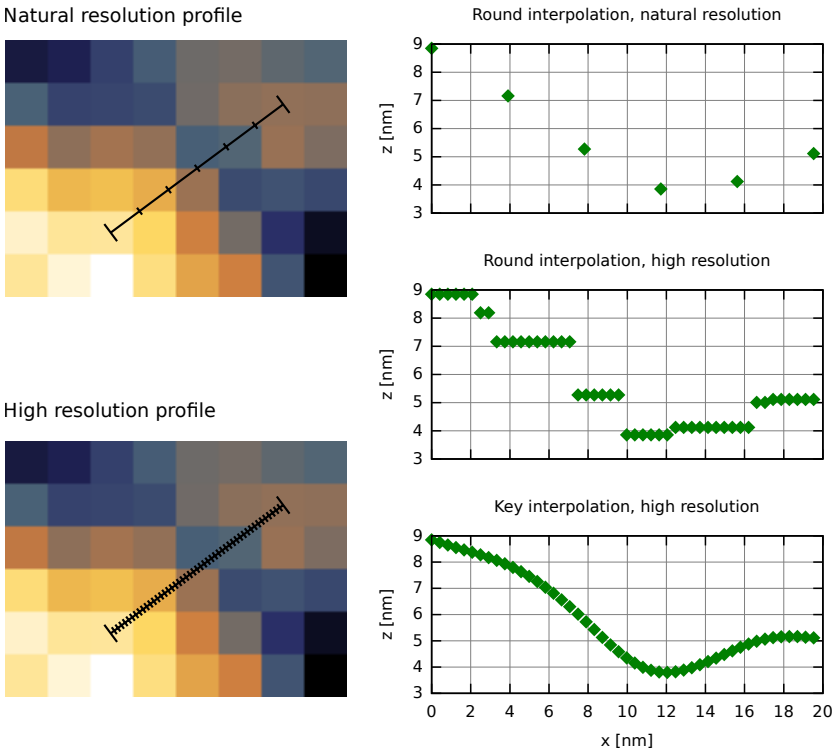


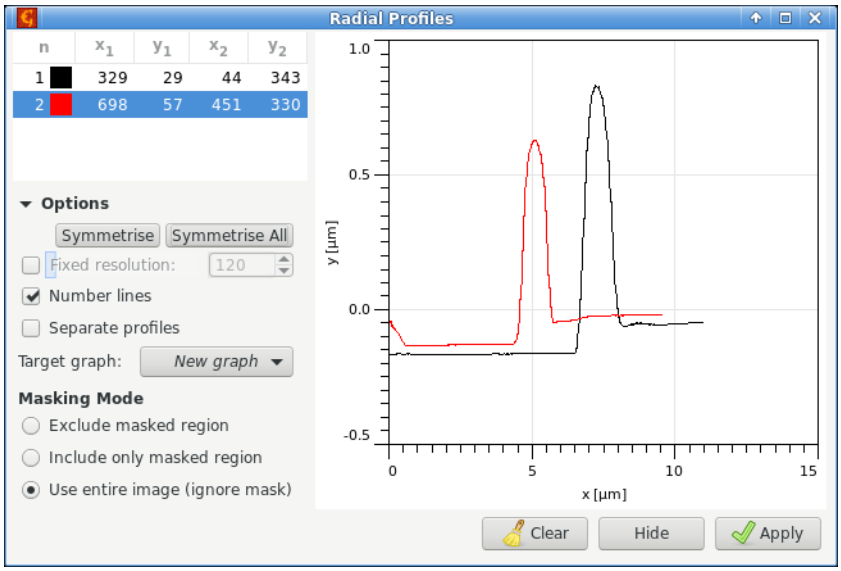
Illustration of data sampling in profile extraction for oblique lines. The figures on the left show the points along the line where the values are read for natural and very high resolution. The graphs on the right show the extracted values. Comparison of the natural and high resolution profiles taken with Round interpolation reveals that indeed natural-resolution curve points form a subset of the high-resolution points. The influence of the interpolation method on values taken in non-grid positions is demonstrated by the lower two graphs, comparing Round and Key interpolation at high resolution.

In the measurement of profiles across edges and steps, it is often important to choose the profile direction perpendicular to the edge. The buttons *Improve Direction* and *Improve All* can help with this. The first attempts to improve the orthogonality of the currently edited line, while the second tries to improve all selected lines. The line centres are preserved; only the directions of the profiles are adjusted. The automatic improvement is not infallible, but it usually works well for reasonably clean standalone edges.

Radial profiles

The Radial profile tool is somewhat similar to the standard profile tool. However, it does not extract profiles along lines. Instead, the line is used to define a circular area, with the centre marked by a tick. The extracted profile is then obtained by angular averaging. This is useful for extraction of the shapes of symmetrical surface features. The abscissa of the extracted graph is the distance from the centre instead of the distance along the line.

Although the line can be adjusted manually, finding the best centre for the radial profile manually may be difficult. Therefore, the tool can attempt a precise location of the best centre itself. You only need to select the line approximately and then press *Symmetrize* to adjust the currently edited line or *Symmetrize All* to adjust all lines. The lines will be shifted slightly to minimise the differences between line profiles taken in different directions from the centre.



Radial profiles tool with two extracted selected profiles and expanded options.

Conversion to other data types

Function *Data Process* → *Basic Operations* → *Volumize* creates volume data from an image. The height field is interpreted as the surface of a solid object, as usual in AFM. Voxels below the surface (inside the material) are filled with 1s while voxels above the surface (outside) are filled with 0s. The z-coordinate of the volume data therefore corresponds to the image values, while the volume data values are unitless.

Function *Data Process* → *Basic Operations* → *Volumize Layers* creates volume data from a sequence of images. All images in the file must have the same dimensions. They are then treated as planes in the volume data that are created by stacking the images. The z-coordinate of the volume data therefore corresponds to the stack index (and can be specified in the dialogue), while the volume data values have the same units as the image data.

Function *Data Process* → *Basic Operations* → *XYZize* creates XYZ data from an image. Each image pixel corresponds to one point in the created XYZ data. Therefore, the xy coordinates thus form a regular grid and all units are the same as for the image.

Function *Data Process* → *Basic Operations* → *XYZ from Channels* creates XYZ data from three images, one containing x coordinates, one y and one z. This is useful when the data were measured nominally on a regular grid, but the actual measurement locations differed from regular grid points and their coordinates were recorded in additional channels.

4.2 Interpolation

Most geometrical transformations, such as rotation, scaling or drift compensation utilize or depend on data interpolation. Also some other operations, e.g. profile extraction, can work with values between individual pixels and hence involve interpolation. Since SPM data are relatively coarsely sampled compared to measured details (full images are typically only a few hundred pixels in width and height), the interpolation method used can become critical for proper quantitative analysis of data properties. Gwyddion implements several interpolation methods [1] and the user can choose which method to use for most of the modules using interpolation.

Here, we describe the principles and properties of one-dimensional interpolation methods. All implemented two-dimensional interpolation methods are separable and thus simply composed of the corresponding one-dimensional methods. The following interpolation methods are currently available:

Round Round interpolation (also called nearest neighbourhood interpolation) is the simplest method – it just takes rounded value of the expected position and finds therefore the closest data value at integer position. Its polynomial degree is 0, regularity C^{-1} and order 1.

Linear Linear interpolation is a linear interpolation between the two closest data values. The value z at point of relative position x is obtained as

$$z = (1-x)z_0 + xz_1$$

where z_0 and z_1 are values at the preceding and following points, respectively. Its polynomial degree is 1, regularity C^0 and order 2. It is identical to the second-order B-spline.

Key Key interpolation (more precisely Key's interpolation with $a = -1/2$ which has the highest interpolation order) makes use also of values in the before-preceding and after-following points z_{-1} and z_2, respectively. In other words it has support of length 4. The value is then obtained as

$$z = w_{-1}z_{-1} + w_0z_0 + w_1z_1 + w_2z_2$$

where

$$\begin{aligned} w_{-1} &= \left(-\tfrac{1}{2} + (1 - \tfrac{x}{2})x\right)x \\ w_0 &= 1 + (-\tfrac{5}{2} + \tfrac{3}{2}x)x^2 \\ w_1 &= \left(\tfrac{1}{2} + (2 - \tfrac{3}{2}x)x\right)x \\ w_2 &= (-\tfrac{1}{2} + \tfrac{x}{2})x^2 \end{aligned}$$

are the interpolation weights. Key's interpolation degree is 3, regularity C^1 and order 3.

Schaum Schaum interpolation (more precisely fourth-order Schaum) has also support of length 4. The interpolation weights are

$$\begin{aligned} w_{-1} &= -\tfrac{1}{6}x(x-1)(x-2) \\ w_0 &= \tfrac{1}{2}(x^2-1)(x-2) \\ w_1 &= -\tfrac{1}{2}x(x+1)(x-2) \\ w_2 &= \tfrac{1}{6}x(x^2-1) \end{aligned}$$

Its polynomial degree is 3, regularity C^0 and order 4.

NNA Nearest neighbour approximation is again calculated from the closest four data values but unlike all others it is not piecewise-polynomial. The interpolation weights are

$$w_k = \frac{\dfrac{1}{r_k^4}}{\displaystyle\sum_{j=-1}^{2} \frac{1}{r_j^4}},$$

for $k = -1, 0, 1, 2$, where $r_{-1} = 1 + x$, $r_0 = x$, $r_1 = 1 - x$, $r_2 = 2 - x$. Its order is 1.

B-spline The weights are

$$\begin{aligned} w_{-1} &= \tfrac{1}{6}(1-x)^3 \\ w_0 &= \tfrac{2}{3} - x^2(1 - \tfrac{x}{2}) \\ w_1 &= \tfrac{1}{6} + \tfrac{1}{2}x\big(1 + x(1-x)\big) \\ w_2 &= \tfrac{1}{6}x^3 \end{aligned}$$

However, they are not used with directly function values as above, but with interpolation coefficients calculated from function values [1]. Its polynomial degree is 3, regularity C^2 and order 4.

O-MOMS The weights are

$$\begin{aligned} w_{-1} &= \tfrac{4}{21} + \left(-\tfrac{11}{21} + (\tfrac{1}{2} - \tfrac{x}{6})x\right)x \\ w_0 &= \tfrac{13}{21} + \left(\tfrac{1}{14} + (-1 + \tfrac{x}{2})x\right)x \\ w_1 &= \tfrac{4}{21} + \left(\tfrac{3}{7} + (\tfrac{1}{2} - \tfrac{x}{2})x\right)x \\ w_2 &= (\tfrac{1}{42} + \tfrac{1}{6}x^2)x \end{aligned}$$

However, they are not used directly with function values as above, but with interpolation coefficients calculated from function values [1]. Its polynomial degree is 3, regularity C^0 and order 4.

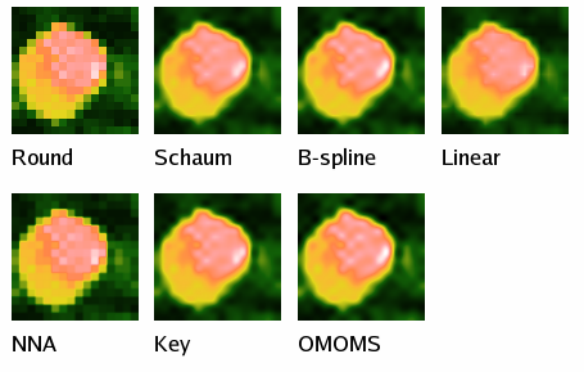


Illustration of the available interpolation types (the original pixels are obvious on the result of Round interpolation). All images have identical false colour map ranges.

References

[1] P. Thévenaz, T. Blu, M. Unser: Interpolation revisited. IEEE Transactions on medical imaging 10 (2000) 739–758, doi:10.1109/-42.875199

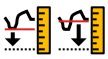
4.3 Data Levelling and Background Subtraction

Levelling

The data obtained from SPM microscopes are very often not levelled at all; the microscope directly outputs raw data values computed from piezoscanner voltage, strain gauge, interferometer or other detection system values. This way of exporting data enables the user to choose his/her own method of levelling data.

The choice of levelling method should be based on your SPM system configuration. Basically, for systems with independent scanner(s) for each axis, plane levelling should be sufficient. For systems with scanner(s) moving in all three axes (tube scanners) 2nd degree polynomial levelling should be used.

Of course, you can use higher degree levelling for any data, however, this can suppress real features on the surface (namely waviness of the surface) and therefore alter the statistical functions and quantities evaluated from the surface.

Fix Zero and Zero Mean Value 

Data Process → *Level* → *Fix Zero*

Data Process → *Level* → *Zero Mean Value*

The simplest functions that are connected with data levelling are *Fix Zero* and *Zero Mean Value* that just add a constant to all the data to move the minimum and mean value to zero, respectively.

Plane Level 

Data Process → *Level* → *Plane Level*

Plane levelling is usually one of the first functions applied to raw SPM data. The plane is computed from all the image points and is subtracted from the data.

If a mask is present plane levelling offers to use the data under mask for the plane fitting, exclude the data under mask or ignore the mask and use the entire data.

Tip You can quickly apply plane levelling by simply right-clicking on the image window and selecting *Level*.

Three Point Levelling Tool 

The Three Point Levelling tool can be used for levelling very complicated surface structures. The user can simply mark three points in the image that should be at the same level, and then click *Apply*. The plane is computed from these three points and is subtracted from the data.

Facet Level 

Data Process → Level → Facet Level

Facet Level levels data by subtracting a plane similarly to the standard Plane Level function. However, the plane is determined differently: it makes facets of the surface as horizontal as possible. Thus for surfaces with flat horizontal areas it leads to much better results than the standard Plane Level especially if large objects are present.

On the other hand, it is not suitable for some types of surface. These includes random surfaces, data with considerable fine noise and non-topographic images as the method does not work well if the typical lateral dimensions and "heights" differ by many orders.

Similarly to Plane Level, Facet Level can include or exclude the data under mask. This choice is offered only if a mask is present.

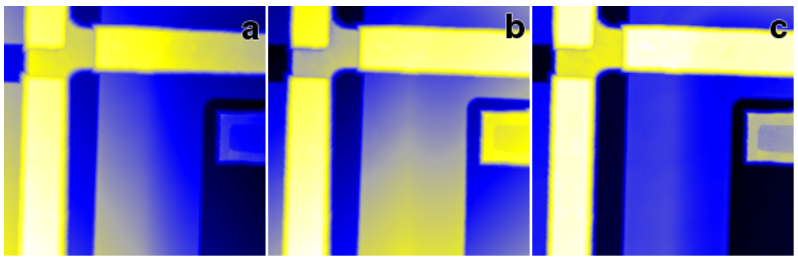
Finding the orientation of the facets is an iterative process that works as follows. First, the variation of local normals is determined:

$$\beta^2 = \frac{1}{N}\sum_{i=1}^{N} \mathbf{n}_i^2$$

where $\mathbf{n}_i$ is the vector of local facet normal (see inclination coordinates) in the i-th pixel. Then the prevalent normal is estimated as

$$\mathbf{n} = \frac{\sum_{i=1}^{N} \mathbf{n}_i \exp\left(-c\frac{\mathbf{n}_i^2}{\beta^2}\right)}{\sum_{i=1}^{N} \exp\left(-c\frac{\mathbf{n}_i^2}{\beta^2}\right)}$$

where $c = 1/20$ is a constant. Subsequently, the plane corresponding to the prevalent normal $\mathbf{n}$ is subtracted and these three steps are repeated until the process converges. The gaussian weighting factors serve to pick a single set of similar local facet normals and converge to their mean direction. Without these factors, the procedure would obviously converge in one step to the overall mean normal – and hence would be completely equivalent to plain plane levelling.



Facet Level example: (a) uncorrected, sloping data; (b) data levelled by standard plane fitting (Plane Level); (c) data levelled by Facet Level.

Level Rotate

Data Process → Level → Level Rotate

Level Rotate behaves similarly to Plane Level, however it does not simply subtract the fitted plane from the data. Instead, this module takes the fitted plane parameters and rotates the image data by a calculated amount to make it lie in a plane. So unlike Plane Level, this module should therefore preserve angle data in the image.

Background Subtraction

Gwyddion has several special modules for background subtraction. All allow you to extract the subtracted background to a separate data window.

Tip For finer control, you can use any of Gwyddion's filtering tools on an image, and then use the Data Arithmetic module to subtract the results from your original image.

Polynomial Background 

Data Process → Level → Polynomial Background

Fits data by a polynomial of the given degree and subtracts this polynomial. In the *Independent degree* mode the horizontal and vertical polynomial orders can be generally set separately, i.e. the fitted polynomial is

$$\sum_{j=0}^{m}\sum_{k=0}^{n} a_{j,k}x^j y^k$$

where m and n are the selected horizontal and vertical polynomial degrees, respectively. In the *Limited total degree* mode the fitted polynomial is

$$\sum_{j+k\le n} a_{j,k}x^j y^k$$

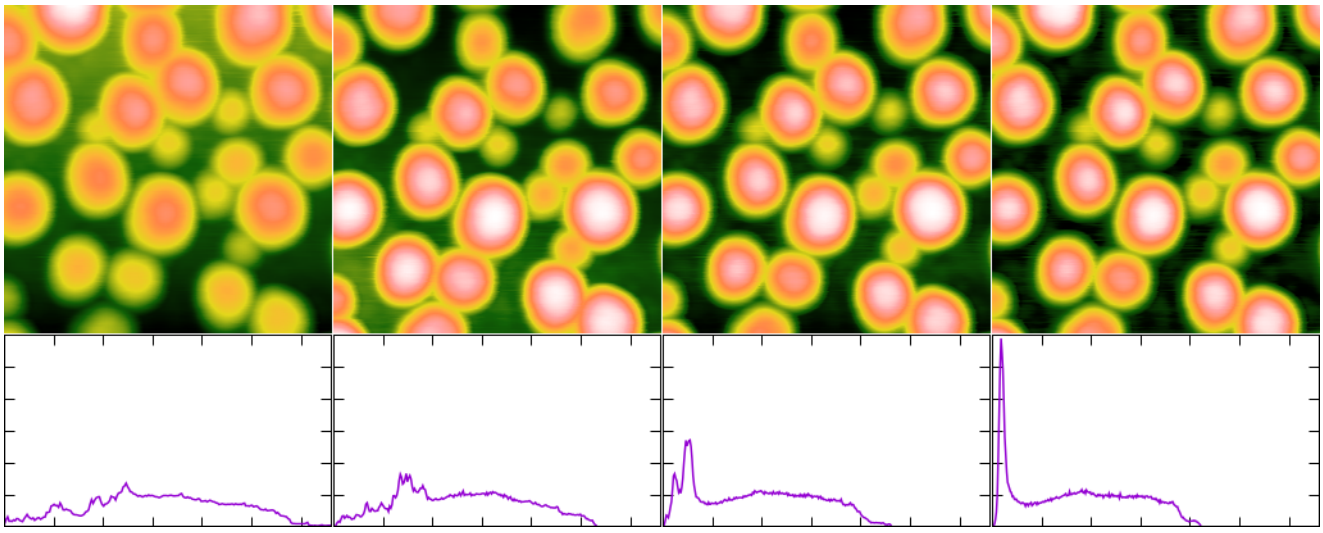
where n is the selected total polynomial degree.

Similarly to Plane Level, polynomial background subtraction can include or exclude the data under mask. This choice is offered only if a mask is present.

Flatten base 

Data Process → Level → Flatten Base

When a number of large features are present on a flat base surface the combination of masking, plane, facet and/or polynomial levelling can be used to level the flat base. It can require, however, several steps and trial and error parameter adjustment. *Flatten Base* attempts to perform this levelling automatically using a combination of facet and polynomial levelling with automated masking. It attempts to maximise the sharpness of the height distribution peak corresponding to the flat base surface.



Flatten Base example: original image, levelled using Facet Level, levelled using Plane Level, levelled using Flatten Base. The original image is shown in linear colour scale, the levelled images are shown in adaptive colour scale. The graph below each image shows the corresponding height distribution (with the same axis ranges).

Revolve Arc 

Data Process → Level → Revolve Arc

Data are levelled by revolving a virtual "arc" of given radius horizontally or vertically over (or under) the data. The envelope of this arc is treated as a background, resulting in removal of features larger than the arc radius (approximately). It is also possible to apply the levelling in both directions – in this case the arc is first revolved horizontally, then vertically.

By default the arc is revolved on the bottom side, preserving positive features (peaks). It is also possible to roll it on the top side by enabling *Invert height*.

Revolve Sphere 

Data Process → *Level* → *Revolve Sphere*

Data are levelled by revolving a virtual "sphere" of given over (or under) the data. The envelope of this sphere is treated as a background, resulting in removal of features larger than the sphere radius (approximately).

Median Level 

Data Process → *Level* → *Median Level*

The median levelling operation filters data with a median filter using a large kernel and treats the result as background. Only features smaller than approximately the kernel size will be kept.

Trimmed Mean

Data Process → *Level* → *Trimmed Mean*

A more general variant of Median Level is the trimmed mean filter. In trimmed mean given fractions of lowest and highest values are discarded and the remaining data are averaged. For no discarded values the filter is identical to the mean value filter, whereas for the maximum possible number of discarded values (symmetrically) it becomes the median filter. The amount of discarded values can be given as a fraction (percentile) or as a specific number.

K-th Rank Filter

Data Process → *Integral Transforms* → *Rank Filter*

Another operation which generalises Median Level is the k-th rank filter. Imaging all the values in the neighbourhood of a pixel sorted, the median filter selects the value exactly in the middle. The k-th rank filter allows specifying any other rank or percentile to select at the filter output.

This means the filter can reproduce minimum, maximum and median filters – and anything between. It can also directly perform two rank filters simultaneously and calculating their difference. This is not useful for levelling, but allows for instance the calculation of local inter-quartile range for each pixel.

4.4 Masks

The mask-related functions can be divided into three main groups:

Creation Masks are created by various types of marking functions, namely grain marking functions (Mark by Threshold, Mark by Watershed), defect marking functions (Mask of Outliers, Mark Scars) and feature marking functions (Mask by Correlation, Facet Analysis, Certainty Map). In addition, some general mask editing functions provide means to create masks from scratch.

Masks are also used to mark invalid pixels in files imported from formats that distinguish between valid and invalid pixels since Gwyddion does not have a concept of invalid pixels.

Application In general, the mask-covered area is considered to be the area of interest, i.e. the area to operate on. This applies namely to statistical functions such as the Statistical Quantities tool. Function Interpolate Data Under Mask replaces the data under mask, while the Remove Grains tool can perform such replacement for individual grains. There are several functions for the examination of grain properties, see section Grain Statistics.

Some functions ask whether to consider the area under mask included or excluded (or ignore the mask), namely levelling functions. Such choice is offered only if a mask is present on the data.

Editing A few basic mask operations, such as inversion or complete removal, are available in *Data Process* → *Mask* menu. More advanced functions include the grain-oriented Remove Grains tool and Grain Filter that provide different means to remove parts of the mask, as well as Mask Editor tool and Mark With focused on general mask editing.

This section describes general mask functions that are not tied to specific applications.

Basic operations

Inversion and Removal 

Basic operations in the *Data Process* → *Mask* menu include *Invert* that inverts the mask, making marked pixels unmarked and vice versa, and *Remove* that removes the mask from the data. There functions are also available in the Mask Editor tool described below. In addition, mask can be removed using the data window context menu.

Extract 

Function *Extract* creates a new image from the mask of the current image. The image will have values 1 in masked pixels and 0 in unmasked pixels.

Shift 

Mask can be shifted horizontally and/or vertically using *Data Process* → *Mask* → *Shift*. After shifting the mask, some its pixels correspond to outside of the image. The *Exterior type* controls how they are filled. The periodic, mirror and border-copying methods work the same as in Extend. In addition all the outside pixels can be set to masked (*Filled*) or unmasked (*Empty*).

Distribute 

Occasionally it is useful to copy a mask to many images. This can be done using the *Distribute* function. It can copy the mask either to all images within one file or all images in all open files. If the checkbox *Preserve existing masks* is enabled the function avoids replacing existing masks. Note that the mask can be copied only to images with the same pixel dimensions. Hence images with different dimensions are automatically excluded.

Mask Editor Tool 

The Mask Editor tool is the "Swiss Army knife" for mask modification. It provides two groups of functions: editing of the mask by drawing directly in the data window and global operations with the mask such as inversion or growing and shrinking.

The direct mask modification is controlled by the buttons in the *Editor* group. It can be done in two ways: by selecting geometrical shapes that are subsequently filled or erased (using the *Shapes* option), and by freehand drawing operations using drawing tools (option *Drawing Tools*).

Buttons in the *Mode* row select how the geometrical shape drawn in the data window will modify the mask:

Set  The mask is set to the drawn shape, discarding any mask already present.

Add  The mask is extended by the drawn shape (if there is no mask yet a mask is created).

Subtract  The drawn shape is cut out from the mask This function has no effect if there is no mask.

Intersect  The mask is set to the intersection of the drawn shape and the already present mask. This function has no effect if there is no mask.

Buttons in the *Shape* row control which shape is drawn on the mask. The choices include rectangles, ellipses and thin lines.

Freehand drawing tools are selected by buttons in the *Tool* row:

Pencil  Freehand drawing with a pencil of radius specified by parameter *Radius*. Note this may be slow on slow computers and/or large data fields.

Eraser  Freehand erasing with an eraser of radius specified by parameter *Radius*. Note this may be slow on slow computers and/or large data fields.

Fill  Bucket-filling of a contiguous unmasked area.

Erase-fill  Bucket erase-filling of a contiguous masked area.

The basic global operation with masks, i.e. inversion, removal and filling the entire data field area with a mask are available in the *Actions* row. Additional operations include:

Grow  Extends the mask by *Amount* pixels on each side. More precisely, the mask is extended by one pixel on each side and this is repeated *Amount* times.

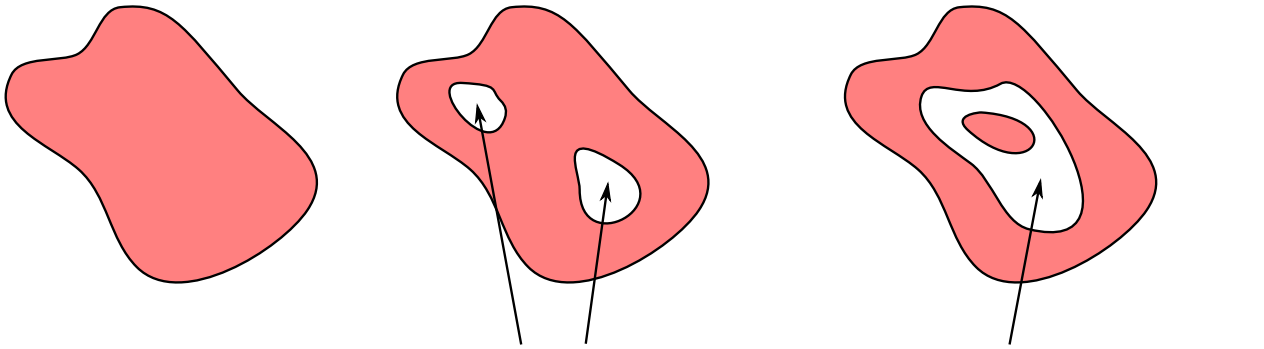
Normally, growing does not distinguish between individual parts of the mask. Parts that grow so much that they touch therefore merge. This can be prevented by *Prevent grain merging by growing* which makes individual parts of the mask stop growing once there is only one-pixel space between them.

When the mask is extended by a large amount it can become important how exactly we measure the distance from the mask edge because it essentially determines the shape to which small grains will grow. This is controlled by option *Distance type*. The various distance types are described in paragraph Distance Transform. Usually, the Euclidean distance is preferable.

Shrink Reduces the mask by *Amount* pixels from each side. More precisely, the mask is reduced by one pixel from each side and this is repeated *Amount* times.

The reduction may or may not occur from the data field borders. This is controlled by the *Shrink from border* check box. Similar to growing, it is possible to specify the distance type also for shrinking.

Fill Voids Makes the grains simply-connected, i.e. without any holes, by filling the holes in grains. The checkbox *Fill non-simple-connected* controls if only simple holes are filled, i.e. holes that are themselves simply-connected, or all holes are filled, regardless of topology.



Explanation of voids connectivity for Fill Voids. *Without* Fill non-simple-connected *only voids in the middle mask would be filled. With this option enabled all voids would be filled.*

Mark With

Data Process → *Mask* → *Mark With*

Mark With can create or modify masks using another mask or data of the same dimensions. The operations that can be applied to the current mask are the same as in the Mask Editor tool: creation, union, subtraction and intersection. The source of the other mask can be one of the following:

Mask This is the simplest case, a mask can be combined with another mask using the specified logical operations.

Data In the Data mode, another height field is used as the other mask source. The mask consists of pixels within a range of heights, specified as relative values within the total range. To use pixels outside a certain range for the masking, set the upper bound to a smaller value than the lower bound.

Presentation The Presentation mode differs from Data mode only in that a presentation is used instead of the data.

This is an exception to the rule stating that presentations are never used for further processing. Sometimes it can be useful to mark, for instance, edges found on the data even though the corresponding presentation visualizes a quantity weird from the physical point of view.

Morphological Operation

Classical morphological operations can be performed with masks using *Data Process* → *Mask* → *Morphological Operation*. All the operations take the mask as one operand and so-called structuring element as the second operand that determines the precise effect. The structuring element can be chosen from a few predefined basic shapes (disc, octagon, square, diamond) or it can be given as another mask.

The basic shapes are all symmetrical. When another mask is used as the structuring element it may be asymmetrical or have empty borders. In this case the structuring element has the full dimensions of the selected mask image, including any empty

borders. This full-size image is considered centred. If you want to remove empty borders from the structuring elements mask, enable the option *Trim empty border*. The structuring element will be then again centred but after the empty border removal.

The available operations include:

Erosion Erosion removes small features and reduces the size or larger features – by amount and shape given by the structuring element. More precisely, if the structuring element is placed to a pixel of the mask and it is entirely contained in the mask then such pixel is kept in the mask. All other mask pixels are cleared.

Dilation Dilation fills small holes and increases the size of features – by amount and shape given by the structuring element. More precisely, if the structuring element is rotated by 180 degrees and placed to each pixel of the mask the result of dilation is the union of all these shifted structuring elements.

Opening Opening is erosion followed by dilation. The dilation corrects the effect of the preceding erosion where the mask conforms to the structuring elements. The net effect is that small non-conforming bumps and grains are removed but conforming parts are kept intact. The mask shape is thus cleaned and simplified.

Closing Closing is dilation followed by erosion. The erosion corrects the effect of the preceding dilation where the mask conforms to the structuring elements. The net effect is that small non-conforming holes and dents are removed but conforming parts are kept intact. The mask shape is thus simplified.

ASF Opening A simple opening or closing filter may be too rough for correction of shape flaws of varying sizes. The alternating sequential filter consists of a sequence of closing and opening operations with progressively increasing structuring elements up to the given size and preserves better the shape of features that should not be removed by the opening. The final operation of this filter is the opening.

ASF Closing A simple opening or closing filter may be too rough for correction of shape flaws of varying sizes. The alternating sequential filter consists of a sequence of opening and closing operations with progressively increasing structuring elements up to the given size and preserves better the shape of features that should not be removed by the closing. The final operation of this filter is the closing.

Distance Transform 

The distance transform assigns to each pixel its distance to the mask boundary. It is available as *Data Process* → *Mask* → *Distance Transform*. This operation is, in a certain sense, complementary to the watershed. The distance transform can perform the transform using true Euclidean distance

$$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

as well as classical simple distances such as the city-block (4-neighbourhood)

$$|x_1-x_2|+|y_1-y_2|$$

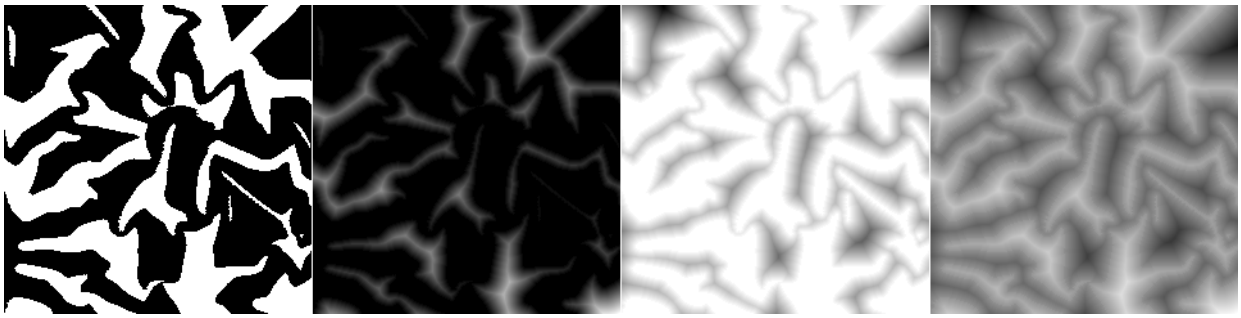
and chessboard (8-neighbourhood)

$$\max(|x_1-x_2|,|y_1-y_2|)$$

distances that are useful mainly for didactic purposes. In addition, octagonal distances are available that are calculated by taking the city-block and chessboard distance steps alternatively, obtaining '48' and '84' variants depending on which step is taken first. Finally, an octagonal distance without any specifier is available that is the average of the two variants.

If the transform is applied to grain interiors the distance is zero outside grains and increases towards the grain 'centres'. Conversely, it can be applied to the exteriors and it is then highest for pixels farthest from any grain. It is also possible to calculate signed two-side transform which is the difference of the two transforms, i.e. it is positive inside grains and negative outside.

Option *Shrink from border* controls the handling of borders. When it is enabled, image boundaries are considered to be also grain (or non-grain) boundaries. Pixels on the image edge thus cannot receive large values. When it is disabled, the grains (or non-grains) are effectively infinitely large outside the image. So pixels close to the boundary can receive large distance values.

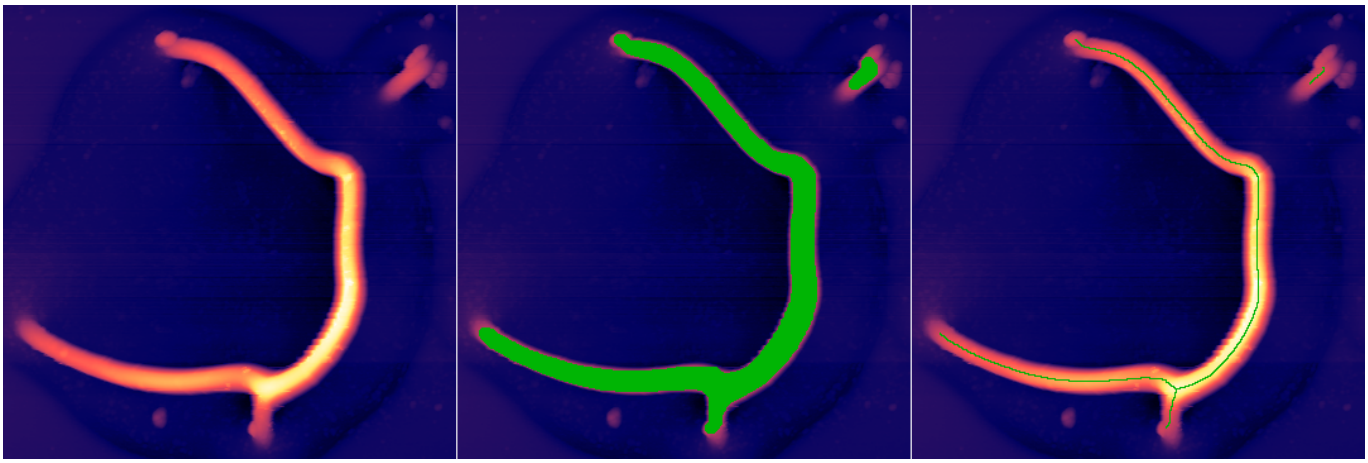


Euclidean distance transform example: the source mask shown as white on black, interior distance transform, exterior transform (with inverted greyscale for comparison) and the signed transform.

Thinning

Thinning is another classical morphological operation, available as *Data Process* → *Mask* → *Thin*. It reduces the mask to a skeleton consisting only of thin lines characterizing the overall shape. If the mask boundary is jagged thinning can result in a large number of extra strands. Therefore it is often useful to apply a simplifying operation beforehand (opening or closing).

It should be noted that this function works only with the mask. It does not use any information from the underlying height field. Therefore, even though the thinned mask can often coincide with summits and ridges of marked topographical features, this is generally just a coincidence.



Mask thinning example: topographical image of buckling of a polymer film (left); the same image with a mask covering the bump on the surface (centre); and the same mask after thinning (right).

Noisify

Function *Data Process* → *Mask* → *Noisify* adds random salt and/or pepper noise to the mask, as selected by the noise type (symmetrical, positive or negative). This is useful mainly for the evaluation and development of data processing methods.

Parameter *Density* determines the fraction of image pixels touched. If *Alter only boundaries* is enabled a mask pixels is not changed unless it is currently at the boundary, i.e. some of its 4-neighbours differs from the pixel.

4.5 Filters

Basic Filters Tool

The basic Filters tool lets you apply several simple filters to your image. This can be very useful for data denoising; however, the real measured data will get altered in the process, so great care should be taken not to destroy important features of the image.

All filters alter the data based on the values in the neighbourhood of the filtered pixel. For some filters the size and shape of the neighbourhood is fixed. For most, however, the size can be controlled with parameter *Size*. Apart from Gaussian-based filters (described below) the size determines the diameter of the neighbourhood in pixels. It is thus advisable to use filters of odd sizes because their effect is symmetric.

Mean value The filtered value is the mean value of the neighbourhood of the original pixel.

Median value The filtered value is the median value of the neighbourhood of the original pixel.

Conservative denoise The filter checks whether the value is the extreme within its neighbourhood. Non-extreme values are preserved. Extreme values are replaced by the second highest (lowest) values.

Minimum Also known as erosion filter, it replaces values by the minimum found in the neighbourhood. Specifically, the erosion is performed with a flat disc of specified diameter.

Maximum Also known as dilation filter, it replaces values by the maximum found in the neighbourhood. Specifically, the dilation is performed with a flat disc of specified diameter.

Opening Opening is erosion followed by dilation. Positive bumps are thus removed from the surface, while smooth areas and negative bumps (holes) are more or less preserved.

Closing Closing is dilation followed by erosion. Negative bumps (holes) are thus removed from the surface, while smooth areas and positive bumps are more or less preserved.

ASF opening A simple opening or closing filter may be too rough for correction of defects of varying sizes. The alternating sequential filter (ASF) consists of a sequence of closing and opening operations with progressively increasing discs up to the given size. It preserves better the shape of features that should not be removed by the opening. The final operation of this filter is opening.

ASF closing The ASF closing filter differs from the ASF opening filter only by the order of the elementary operations. The final operation of this filter is closing.

Kuwahara Kuwahara filter is an edge-preserving smoothing filter.

Dechecker A simple slightly smoothing filter specially designed to remove checker pattern from the image while preserving other details. It is a convolution filter with kernel

$$w_{\text{dechecker}} = \begin{pmatrix} 0 & 1/144 & -1/72 & 1/144 & 0 \\ 1/144 & -1/18 & 1/9 & -1/18 & 1/144 \\ -1/72 & 1/9 & 7/9 & 1/9 & -1/72 \\ 1/144 & -1/18 & 1/9 & -1/18 & 1/144 \\ 0 & 1/144 & -1/72 & 1/144 & 0 \end{pmatrix}$$

Gaussian The Gaussian filter is a smoothing filter, the size parameter determines the FWHM (full width at half maximum) of the Gaussian. The relation between FWHM and σ is

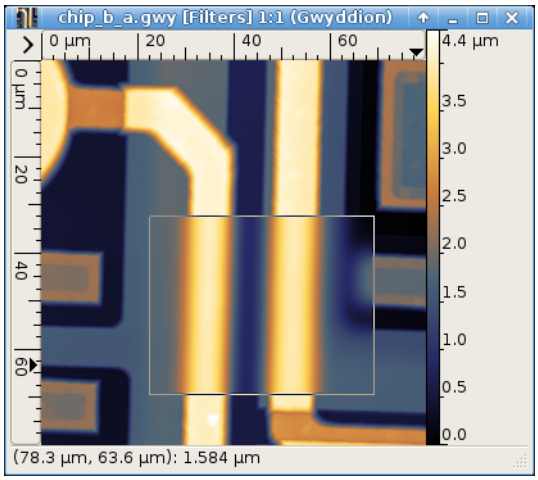
$$\text{FWHM} = 2\sqrt{2\ln 2}\,\sigma \approx 2.35482\sigma$$

Sharpen A simple sharpening operation which is complementary to the Gaussian filter. First it smooths the image further with the corresponding Gaussian filter. The result is then extrapolated backwards to estimate the sharper image.

By default, these filters will be applied to the entire image. However, you can apply a filter to a specific region within your image by selecting it with the mouse. This can be useful for correcting poorly measured areas within a good image.

It is also possible to limit the area by masking. In this case the mask determines the active pixels whose values will be modified by the filter. Pixels in the neighbourhood of the active area can however still enter the calculation passively, i.e. their values can influence the result.

Several of the basic filters can be used for simple denoising and defect removal. Further defect marking and removal functions are described in section Data Editing, see for instance Mask of Outliers and Interpolate Data Under Mask. More advanced denoising functions in Gwyddion, for example DWT denoising and FFT filtering, are described in section Extended Data Editing.



Screenshot of filter tool with median filter applied to a rectangular selection

Convolution Filter 

Data Process → *Integral Transforms* → *Convolution Filter*

Discrete convolutions with arbitrary kernels up to 9×9 can be performed using Convolution Filter (see Image Convolution for convolution of two images).

The *Divisor* entry represents a common factor all the coefficients are divided by before applying the filter. This allows to use denormalized coefficients that are often nicer numbers. The normalization can be also calculated automatically when *automatic* is checked. When the sum of the coefficients is non-zero, it makes the filter sum-preserving, i.e. it normalizes the sum of coefficients to unity. When the sum of the coefficients is zero, the automatic factor is simply let equal to 1.

Since many filters used in practice exhibit various types of symmetry, the coefficients can be automatically completed according to the selected symmetry type (odd, even). Note the completion is performed on pressing **Enter** in the coefficient entry.

In a fresh installation only a sample Identity filter is present (which is not particularly useful as it does nothing). This filter cannot be modified, to create a new filter use the *New* button on the *Presets* page.

4.6 Presentations

Presentation modules do not modify the data, instead, they output their results into a separate layer displayed on top of the original data. The other data processing modules and tools will still operate on the underlying data. To remove a presentation, right-click on the data window, and select *Remove Presentation*.

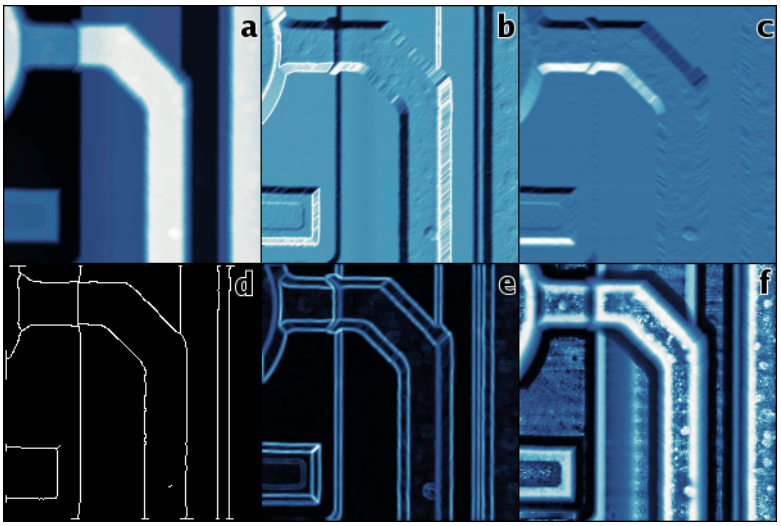
Basic Operations

The *Data Process* → *Presentation* menu contains a few basic presentation operations:

Attach Presentation Attaches another data field as a presentation to the current data. Note that this useful option can be particularly confusing while evaluating anything from the data as all the computed values are evaluated from the underlying data (not from the presentation, even if it looks like the data).

Remove Presentation Removes presentation from the current data window. This is an alternative to the right-click data window menu.

Extract Presentation Extracts presentation from the current data window to a new image in the same file. In this way one can get presentation data for further processing. Note, however, the extracted data have no absolute scale information as presentation often help to visualize certain features, but the produced values are hard or impossible to assign any physical meaning to. Hence the value range of the new image is always $[0, 1]$.



Presentation examples: (a) original data, (b) shading, (c) vertical Prewitt gradient, (d) Canny edge detection, (e) local non-linearity edge detection, (f) local contrast improvement.

Shading Presentation

Data Process → Presentation → Shading

Simple and very useful way of seeing data as illuminated from some direction. The direction can be set by user. It is also possible to mix the shaded and original images for presentational purposes. Of course, the resulting image is meaningless from the physical point of view.

Gradient Detection Presentations

Data Process → Presentation → Gradient

Sobel horizontal and vertical gradient filter and Prewitt horizontal and vertical gradient filter create similar images as shading, however, they output data as a result of convolution of data with relatively standardized kernel. Thus, they can be used for further presentation processing for example. The kernels for horizontal filters are listed below, vertical kernels differ only by reflection about main diagonal.

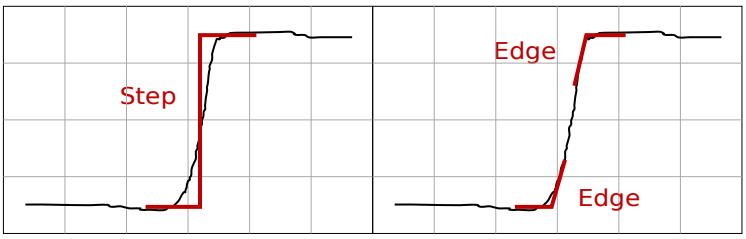
$$w_{\text{Prewitt}} = \begin{pmatrix} 1/3 & 0 & -1/3 \\ 1/3 & 0 & -1/3 \\ 1/3 & 0 & -1/3 \end{pmatrix}, \quad w_{\text{Sobel}} = \begin{pmatrix} 1/4 & 0 & -1/4 \\ 1/2 & 0 & -1/2 \\ 1/4 & 0 & -1/4 \end{pmatrix}$$

Edge Detection Presentations

Data Process → Presentation → Edge Detection

One is often interested in the visualization of the discontinuities present in the image, particularly in discontinuities in the value (zeroth order) and discontinuities in the derivative (first order). Although the methods of location of both are commonly referred to as "edge detection" methods, these are actually quite different, therefore we will refer to the former as to step detection and to the latter as to edge detection. Methods for the detection of more specific features, e.g. corners, are commonly used too, these methods usually are of order zero.

The order of a discontinuity detection method can be easily seen on its output as edge detection methods produce typical double edge lines at value discontinuities as is illustrated in the following figure. While the positions of the upper and lower edge in an ideal step coincide, real-world data tend to actually contain two distinct edges as is illustrated in the picture. In addition, finding two edges on a value step, even an ideally sharp one, is often an inherent feature of edge detection methods.



Step versus edge in one dimension.

The following step and edge detection functions are available in Gwyddion (the later ones are somewhat experimental, on the other hand they usually give better results than the well-known algorithms):

Canny Canny edge detector is a well-known step detector that can be used to extract the image of sharp value discontinuities in the data as thin single-pixel lines.

Laplacian of Gaussians Laplacian presents a simple convolution with the following kernel (that is the limit of discrete Laplacian of Gaussians filter for $\sigma \to 0$):

$$w_{\text{laplace}} = \begin{pmatrix} 0 & 1/4 & 0 \\ 1/4 & -1 & 1/4 \\ 0 & 1/4 & 0 \end{pmatrix}$$

Zero Crossing Zero crossing step detection marks lines where the result of Laplacian of Gaussians filter changes sign, i.e. crosses zero. The FWHM (full width half maximum) of the Gaussians determines the level of details covered. Threshold enables to exclude sign changes with too small absolute value of the neighbour pixels, filtering out fine noise. Note, however, that for non-zero threshold the edge lines may become discontinuous.

Step A step detection algorithm providing a good resolution, i.e. sharp discontinuity lines, and a good dynamic range while being relatively insensitive to noise. The principle is quite simple: it visualizes the square root of the difference between the 2/3 and 1/3 quantiles of the data values in a circular neighbourhood of radius 2.5 pixels centred around the sample.

Gaussian Step Gaussian step is a tunable filter, with trade-off between fine tracing of the steps and resistance to noise. It has one parameter, width of the Gaussian (given as full width at half-maximum). Narrow filters produce fine lines, but also mark local defects. Wide filters distinguish better steps from other features, but the result is more blurred.

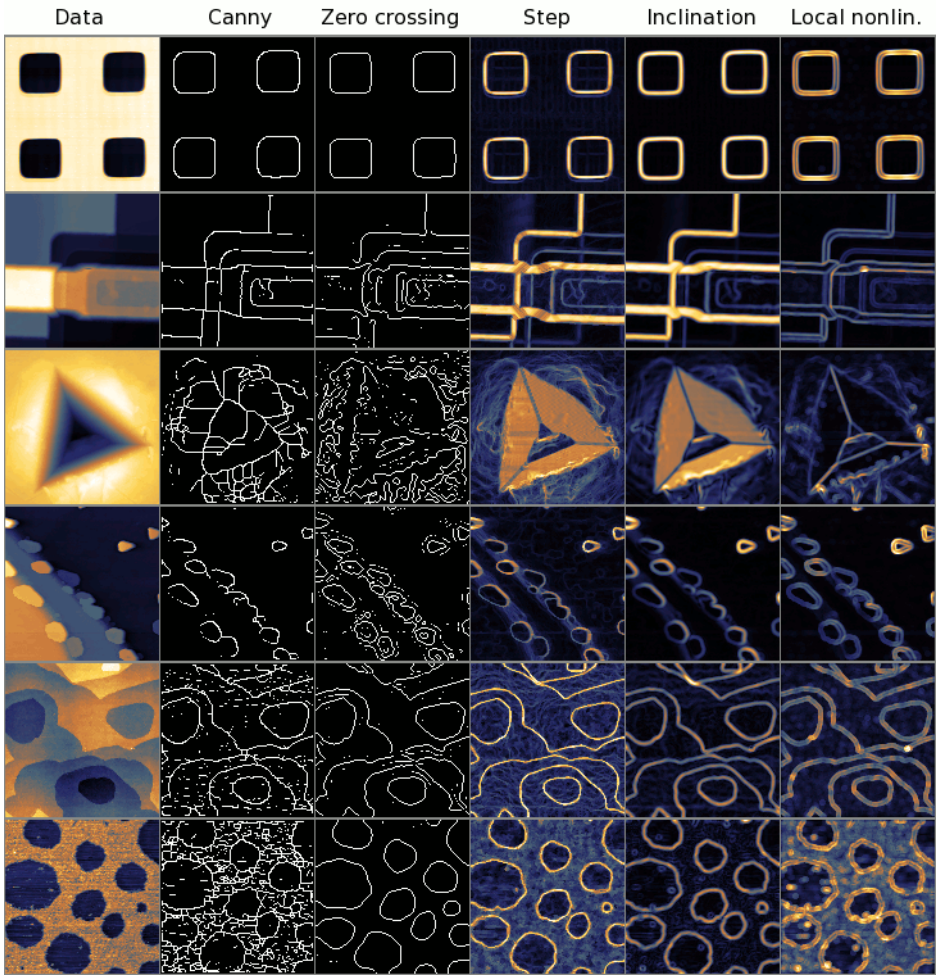
The filter works by convolving the image with kernels consisting of a Gaussian multiplied by sign function, rotated to cover different orientations. The results are then squared and summed together.

RMS This step detector visualizes areas with high local value variation. The root mean square of deviations from the mean value of a circular neighbourhood of radius 2.5 pixels centred around each sample is calculated and displayed.

RMS Edge This function essentially postprocesses RMS output with a filter similar to Laplacian to emphasize boundaries of areas with high local value variation. Despite the name it is still a step detector.

Local Non-Linearity An edge detector which visualizes areas that are locally very non-planar. It fits a plane through a circular neighbourhood of radius 2.5 pixels centred around each sample and then it calculates residual sum of squares of this fit reduced to plane slope, i.e. divided by $1 + b_x^2 + b_y^2$ where b_x and b_y are the plane coefficients in x and y directions, respectively. The square root is then displayed.

Inclination Visualizes the angle ϑ of local plane inclination. Technically this function belongs among step detectors, however, the accentuation of steps in its output is not very strong and it is more intended for easy visual comparison of different slopes present in the image.



Comparison of step and edge detection methods on several interesting, or typical example data. Canny and Zero crossing are step detectors that produce one pixel wide edge lines, Step and Inclination are step detectors with continuous output, Local nonlinearity is an edge detector – the edge detection can be easily observed on the second and third row. Note zero crossing is tunable, it parameters were chosen to produce reasonable output in each example.

Local Contrast

Data Process → Presentation → Local Contrast

A method to visualize features in areas with low and high value variation at the same time. This is achieved by calculation of local value range, or variation, around each data sample and stretching it to equalize this variation over all data.

Rank

Data Process → Presentation → Rank

An alternative local contrast enhancement method. It is an equalising high-pass filter, somewhat complementary to the median filter. Each pixel value is transformed to its rank among all values from a certain neighbourhood. The neighbourhood radius can be specified as *Kernel size*.

The net effect is that all local maxima are equalised to the same maximum value, all local minima to the same minimum value, and values that are neither maxima nor minima are transformed to the range between based on their rank. Since the output of the filter with radius r can contain at most $\pi(r+1/2)^2$ different values (approximately), the filter also leads to value discretisation, especially for small kernel sizes.

Logscale

Data Process → Presentation → Logscale

Logarithmic scale is used for false colours data presentation.

SEM Image

Data Process → *Presentation* → *SEM Image*

The function renders an SEM image-like presentation from a topographical image using the simplest possible Monte Carlo method. For each surface pixel, a number of lines originating from this pixel are chosen with random directions and Gaussian length distribution. The standard deviation of the Gaussian distribution is controlled with the *Integration radius* parameters. If the other end of the line hits free space, the lightness in the origin pixel is increased. If it hits material, i.e. height value below the surface at the endpoint, the lightness is decreased. More precisely, this describes the *Monte Carlo* method. The number of lines can be controlled with the *Quality* parameter. Equivalently, the same intensity can be calculated by direct integration over all pixels within a circular neighbourhood. This corresponds to the *Integration* method.

Since even this simple computation can take a long time, it is useful to consider how its speed depends on the settings. The computation time for *Integration* depends only on the integration radius. The computation time for *Monte Carlo*, on the other hand, depends essentially only on *Quality* (there is also some dependence on the local topography).

4.7 Scan Line Artefacts

This section describes functions for marking and correction of various artefacts in SPM data related to the line by line acquisition. Scan line alignment and correction methods frequently involve heavy data modification which should be avoided if possible. Unfortunately, scan line defects are ubiquitous and, also frequently, correction may be the only way to proceed. Nevertheless, they should be applied with care.

See also section Data Editing and Correction for methods that deal with general local defects.

Align Rows 

Profiles taken in the fast scanning axis (usually x-axis) can be mutually shifted by some amount or have slightly different slopes. The basic line correction function *Data Process* → *Correct Data* → *Align Rows* deals with this type of discrepancy using several different correction algorithms:

Median A basic correction method, based on finding a representative height of each scan line and subtracting it, thus moving the lines to the same height. Here the line median is used as the representative height.

Modus This method differs from *Median* only in the quantity used: modus of the height distribution. Of course, the modus is only estimated because only a finite set of heights is avaiable.

Polynomial The *Polynomial* method fits a polynomial of given degree and subtracts if from the line. For polynomial degree of 0 the mean value of each row is subtracted. Degree 1 means removal of linear slopes, degree 2 bow removal, etc.

Median difference In contrast to shifting a representative height for each line, *Median difference* shifts the lines so that the median of height differences (between vertical neighbour pixels) becomes zero. Therefore it better preserves large features while it is more sensitive to completely bogus lines.

Matching This algorithm is somewhat experimental but it may be useful sometimes. It minimizes a certain line difference function that gives more weight to flat areas and less weight to areas with large slopes.

Facet-level tilt Inspired by Facet level, this method tilts individual scan lines to make the prevalent normals vertical. This only removes tilt; line height offsets are preserved. In one dimension it is generally a somewhat less effective strategy than for images. However, it can correct the tilt in some cases when other methods do not work at all.

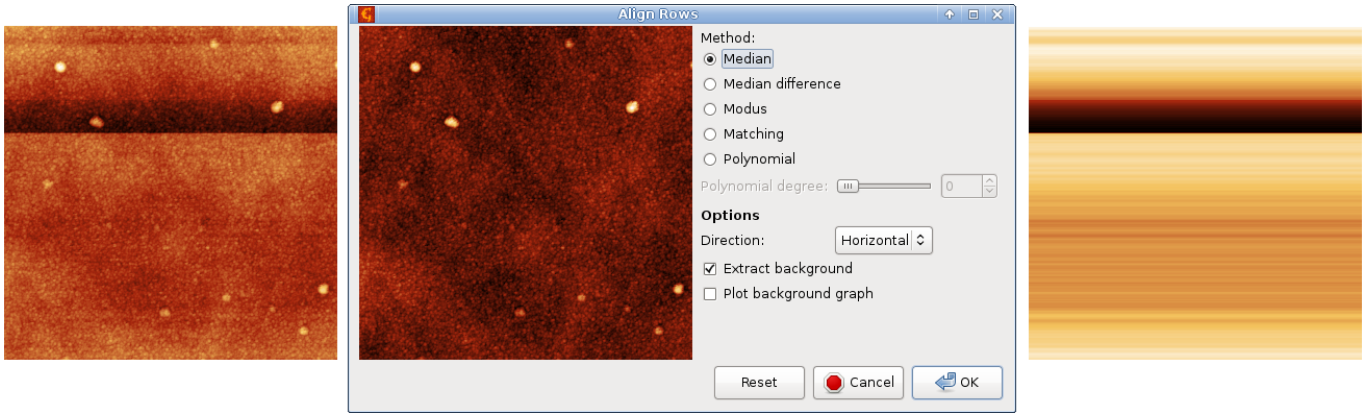
Trimmed mean Trimmed mean lies between the standard mean value and median, depending on how large fraction of lowest and highest values are trimmed. For no trimming (0) this method is equivalent to mean value subtraction, i.e. *Polynomial* with degree 0, for maximum possible trimming (0.5) it is equivalent to *Median*.

Trimmed mean of differences This method similarly offers a continuous transition between *Median difference* and mean value subtraction. It makes zero the trimmed means of height differences (between vertical neighbour pixels). For the maximum possible trimming (0.5) it is equivalent to *Median difference*. Since the mean difference is the same as the difference of mean values (unlike for medians), for no trimming (0) it is again equivalent to *Polynomial* with degree 0.

Similarly as in the two-dimensional polynomial levelling, the background, i.e. values subtracted from individual rows can be extracted to another image. Or plotted in a graph since the value is the same for the entire row.

The line correction function support masking, allowing the exclusion of large features that could distract the correction algorithms. The masking options are offered only if a mask is present though. Note the Path level tools described below offers a different method of choosing the image parts important for alignment. It can be more convenient in some cases.

Tip Use **Ctrl-F** (*Repeat Last*) to run the line correction with the same settings on several images without going through the dialogue.

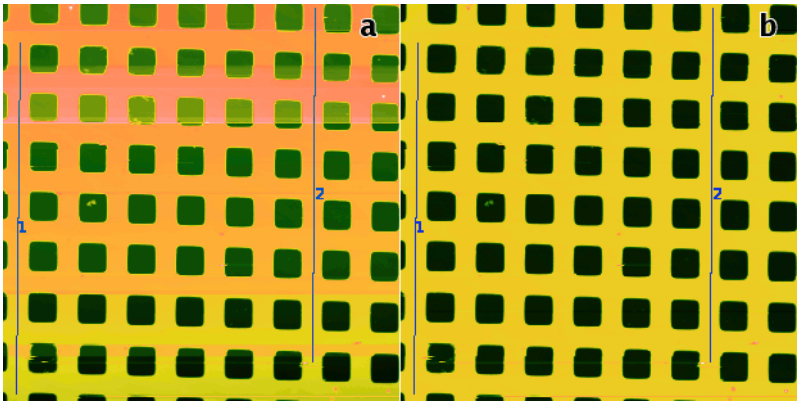


Line correction example: an image with defects (left), the row alignment dialogue previewing the median correction (centre), and extracted row background (right). Note the false colour scales are not the same in the three images.

Path Levelling Tool

The *Path Levelling* tool can be used to correct the heights in an arbitrary subset of rows in complicated images.

First, one selects a number of straight lines on the data. The intersections of these lines with the rows then form a set of points in each row that is used for levelling. The rows are moved up or down to minimize the difference between the heights of the points of adjacent rows. Rows that are not intersected by any line are not moved (relatively to neighbouring rows).



Path Level example: (a) uncorrected data with steps that the automated method may fail to correct, two suitable levelling lines are selected; (b) the result of Path Level application with line width 5.

Step Line Correction

Function *Step Line Correction* attempts to deal with height jumps that may occur in the middle of a scan line. It tries to identify misaligned segments within the rows and correct the height of each such segment individually by subtracting its mean value. Therefore it is often able to correct data with discontinuities in the middle of a row.

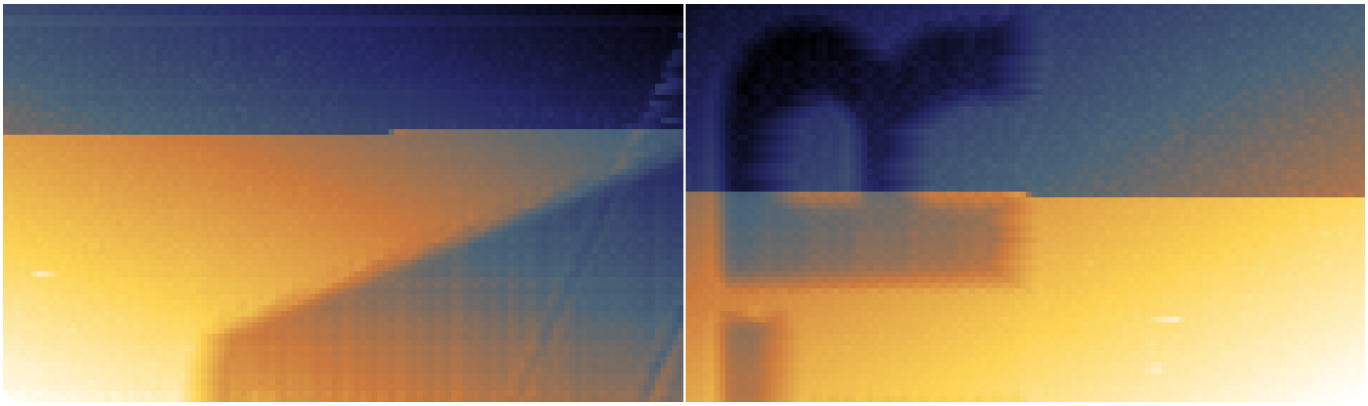
This function is somewhat experimental and the exact way it works can be subject to further changes. Note that it still alignes all rows similarly to the Mean method in Align rows. See Block line correction for a method which attempts to only remove the jumps, preserving everything else.

Block Line Correction

Function *Block Line Correction* also attempts to correct jumps that may occur in the middle of a scan line. It also tries to change the image as little as possible, i.e. less than Step line correction which applies somewhat aggressive correction (and is thus generally more successful at removing the jumps as long as one does not mind the accompanying scan line flattening).

The block line correction identifies the sudden jumps, computes their heights and then shifts height in entire blocks of scan lines between them. That is where the name “block” comes from. In regions where no sudden jump occured mutual the height relations bewtween scan lines are perfectly preserved.

Since entire blocks are shifted the direction of scanning is important in determining which pixels from a contiguous block. It has to be selected as *Scanning direction*. The directions are labelled as for scanning from top to bottom in the slow axis. For scanning from the bottom upwards the meaning is reversed. If the functions seems unable to notice obvious jumps check that the direction is correct.



Step artefacts examples for top-to-bottom scanning in the slow axis. Left: left-to-right. Right: right-to-left.

Threshold determines the sensitivity, in other words how large jumps are considered discontinuities. It is given as a multiple of the root mean square of all differences between two vertical neighbour pixels. By selecting *Marked discontinuities* in *Display* one can get an overview of all locations where the height difference exceeds given threshold. The option *Marked block boundaries* is probably more useful as is shows where a discontinuity spans the entire image width and is, therefore, considered a boundary between two blocks. The number of total detected jumps is displayed below *Threshold*.

Mean Good Profile

Taking mean or median rows from the entire image using Row/Column Statistics tool can provide clean profiles from images formed by repeated scanning of the same feature. This is sometimes done with slow scanning axis disabled entirely, other times with scanning enabled but with much reduced range in the slow axis. If defects are present they should be excluded by masking. When both forward and backward scans are available, comparing them can help determining which parts of the profiles are good and which contain defects.

The *Mean Good Profile* function simplifies this in a couple of common situations:

- If only two images are available, select the other image as *Second image*. A portion of pixels with the highest inter-image differences is then thrown away, as controlled by *Trim fraction*. The rest is used to calculate column-wise mean values, producing the cleaned up mean profile.
- If only one image is available, inter-image comparison is not possible. Therefore, simple column-wise trimmed mean is used instead. *Trim fraction* again controls the portion of pixels thrown away – in this case the fraction applies to each image column independently.

In both modes either the profile graph is shown, or the image with corresponding mask of excluded pixels. The module also shows the variation, i.e. the integral of the absolute value of the derivative. Large variation means more noisy or jagged profile. Hence, lower values usually correspond to better profiles.

Mark Inverted Rows

Function *Mark Inverted Rows* finds scan lines with vertically inverted features and marks them with a mask. Line inversion is an artefact which occasionally occurs for instance in Magnetic Force Microscopy. Since the line is generally only inverted very approximately, value inversion would be a poor correction and one should usually use Laplace’s interpolation for correction.

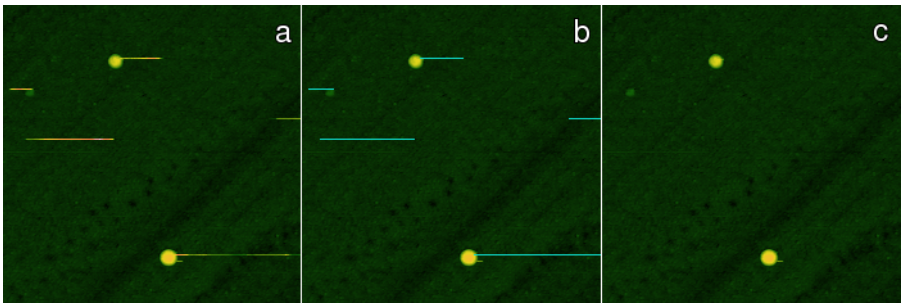
Mark Scars 

Data Process → Correct Data → Mark Scars

Similarly, the `Mark Scars` module can create a mask of the points treated as scars. Unlike Remove Scars which directly interpolates the located defects, this module lets you interactively set several parameters which can fine-tune the scar selection process:

- Maximum width – only scars that are as thin or thinner than this value (in pixels) will be marked.
- Minimum length – only scars that are as long or longer than this value (in pixels) will be marked.
- Hard threshold – the minimum difference of the value from the neighbouring upper and lower lines to be considered a defect. The units are relative to image RMS.
- Soft threshold – values differing at least this much do not form defects themselves, but they are attached to defects obtained from the hard threshold if they touch one.
- Positive, Negative, Both – the type of defects to remove. Positive means defects with outlying values above the normal values (peaks), negative means defects with outlying values below the normal values (holes).

After clicking *Ok* the new scar mask will be applied to the image. Other modules or tools can then be run to edit this data.



Scars marking and removal example: (a) original data with defects, (b) data with marked defects, (c) corrected data.

Remove Scars

Data Process → Correct Data → Remove Scars

Scars (or stripes, strokes) are parts of the image that are corrupted by a very common scanning error: local fault of the closed loop. Line defects are usually parallel to the fast scanning axis in the image. This function will automatically find and remove these scars, using neighbourhood lines to "fill-in" the gaps. The method is run with the last settings used in Mark Scars.

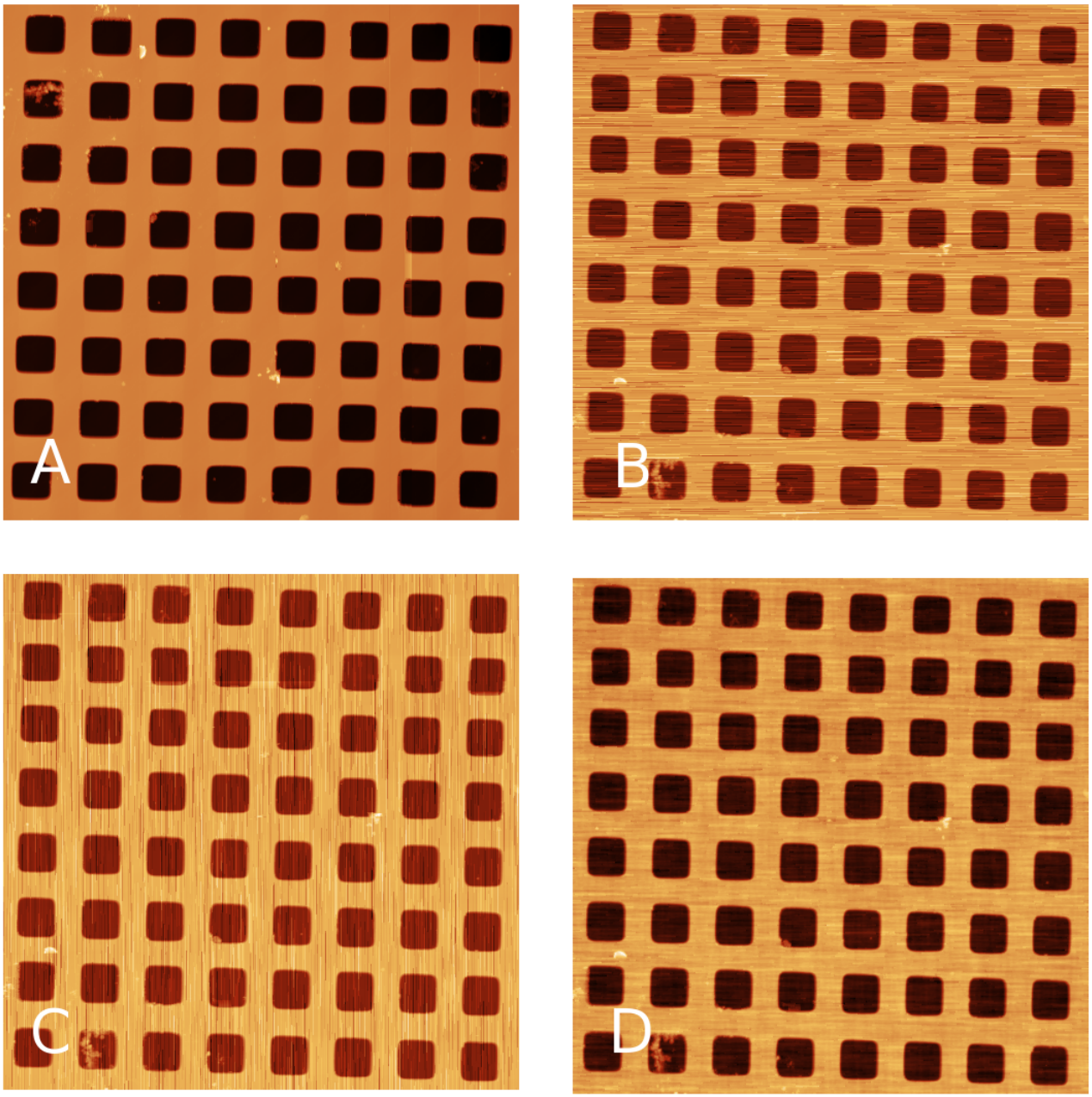
XY Denoising

Data Process → Multidata → XY denoise

The function denoises and image on the basis of two measurements of the same area – one performed in x direction and one in y direction (and rotated back to be aligned the same way as the x-direction one). It is based on work of E. Anguiano and M. Aguilar (see [1]).

The denoising works by performing the Fourier transform of both images, combining the information them in the frequency space, and then using backward Fourier transform in order to get the denoised image. It is useful namely for the removal of large scars and fast scanning axis stripes.

The images do not appear symmetrically in the procedure. By using option *Average denoising directions* one can apply it in both ways and average the results.



XY denoise procedure simulation: A) original data, B) simulated measurement in x axis, C) simulated measurement in y axis, D) denoised image.

References

[1] E. Anguiano and M. Aguilar: A cross-measurement procedure (CMP) for near noise-free imaging in scanning microscopes. Ultramicroscopy, 76 (1999) 47 doi:10.1016/S0304-3991(98)00074-6

4.8 Local Defects

Several modules enable direct or indirect editing of the SPM data. Even though most data processing changes the data in one way or another, this section describes functions and tools designed specifically for correction of local defects. They remove "bad" data from an image, and replace them using some interpolation method.

See also section Scan Line Artefacts for methods that deal with defects related to scanning.

Remove Spots Tool 

The Remove Spots tool can be used for removing very small parts of the image that are considered a scanning error, dust particle or anything else that should not be present in the data. Note that doing so can dramatically alter the resulting statistical parameters of the surface, so be sure not to remove things that are really present on the surface.

While using this tool you can pick up position of the spot to magnify its neighbourhood in the tool window. Then, in the tool window, select a rectangle or ellipse (depending on chosen *Shape*) around the area that should be removed. You can then select one of several interpolation methods for creating data in place of the former "spot":

- *Hyperbolic flatten* uses information from outer boundary pixels of the selected area to interpolate the information inside area.
- *Laplace solver* solves Laplace's equation to calculate data inside area; the outer boundary is treated as virtual source.
- *Pseudo-Laplace* is a faster approximation of true Laplace equation solution. For small areas one can always use the true solver.
- *Fractal correction* uses whole image to determine fractal dimension. Then tries to create randomly rough data that have the same fractal dimension and put them into the area.
- *Fractal-Laplace blend* is a weighted average of fractal interpolation and Laplace solver. The interior is given by fractal interpolation, but close to region edges the Laplace interpolation is mixed in to provide a smoother continuation to the exterior.
- *Zero* simply fills the area with zeros.

Clicking *Apply* will execute the selected algorithm.

Note Spot removal will only work for regions of size 80 × 80 pixels or smaller. To remove larger regions, create a mask using the Mask Editor tool, then use *Data Process* → *Correct Data* → *Interpolate Data Under Mask*.

Remove Grains Tool 

The simple Remove Grains tool removes manually selected connected parts of mask or interpolates the data under them, or possibly both. The part of mask to remove is selected by clicking on it with left mouse button.

The data available interpolation methods are a subset of those offered by Remove Spots tool, in particular Zero, Laplace's equation solution, Fractal correction and their blend.

Interpolate Data Under Mask 

Data Process → *Correct Data* → *Interpolate Data Under Mask*

This function substitutes data under the mask by the solution of solving the Laplace's equation. The data values around the masked areas define the boundary conditions. The solution is calculated iteratively and it can take some time to converge.

Zero Data Under Mask

Data Process → *Correct Data* → *Zero Data Under Mask*

This function simply fills masked data with zeros, which is sometimes less distracting than interpolation.

Fractal Correction 

Data Process → *Correct Data* → *Fractal Correction*

The Fractal Correction module, like the Interpolate Data Under Mask module, replaces data under the mask. However, it uses a different algorithm to come up with the new data: The fractal dimension of the whole image is first computed, and then the areas under the mask are substituted by a randomly rough surface having the same fractal dimension. The root mean square value of the height irregularities (roughness) is not changed by using this module.

Note This calculation can take some time, so please be patient.

Warning Running this module on data that do not have fractal properties can cause really unrealistic results and is strictly not recommended.

Mask of Outliers 

Data Process → *Correct Data* → *Mask of Outliers*

This module creates mask of areas in the data that not pass the 3σ criterion. All the values above and below this confidence interval are marked in mask and can be edited or processed by other modules afterwards, for instance Interpolate Data Under Mask. This outlier marking method is useful for global outliers with values very far from the rest of the data.

Mask of Disconnected 

Data Process → *Correct Data* → *Mask of Disconnected*

Local outliers are values that stick out from their neighbourhood. Function *Mask of Disconnected* marks data that do not seem to belong in the distribution of surrounding values.

The outlier type to mark can be selected as *Positive*, *Negative* or *Both* for values much larger than their neighbours, much smaller and both types simultaneously, respectively. Note that selecting *Both* can mark different areas than if positive and negative outliers were marked separately and the results combined.

The marking proceeds by subtracting a local background from the image and then marking global outliers in the resulting flattened image. Specifically, the local background is obtained by an opening (minimum), closing (maximum) or a median filter of given radius. The radius of the filter is controlled with *Defect radius*. It determines the maximum size of defect that can be found and marked. However, it is often useful to use larger radius than the actual maximum defect size.

Defect marking sensitivity is controlled with option *Threshold*. Smaller values mean more conservative marking, i.e. less values marked as outliers. Larger value means more values marked as outliers.

4.9 Global Distortions

This section presents extended functions for correction (or creation) of global errors, such as low frequencies modulated on the data or drift in the slow scanning axis. Most of the functions described below deal with distortion purely in the xy plane – for instance Drift Compensation or Straighten Path. Some modify the value distribution (Coerce) or perform complex operations involving both coordinates and values.

Drift Compensation 

Data Process → *Distortion* → *Compensate Drift*

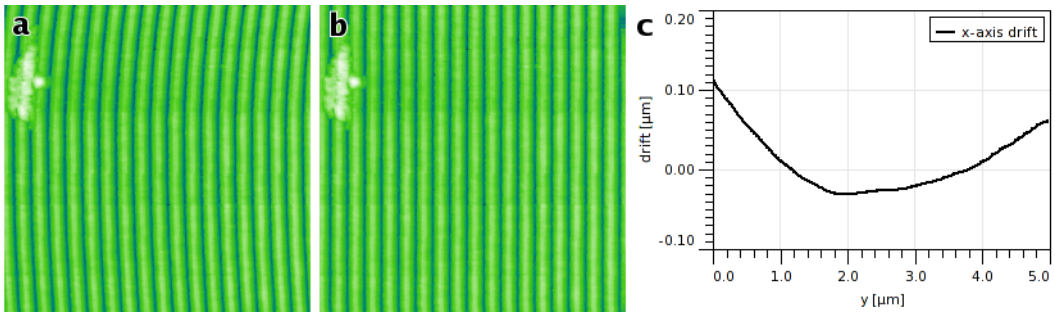
Compensate Drift calculates and/or corrects drift in the fast scanning axis (horizontal). This adverse effect can be caused by thermal effects or insufficient mechanical rigidity of the measuring device.

The drift graph, which is one of possible outputs, represents the horizontal shift of individual rows compared to a reference row (which could be in principle chosen arbitrarily, in practice the zero shift is chosen to minimize the amount of data sticking out of the image after compensation), with the row y-coordinate on the abscissa.

The drift is determined in two steps:

1. A mutual horizontal offset is estimated for each couple of rows not more distant than *Search range*. It is estimated as the offset value giving the maximum mutual correlation of the two rows. Thus a set of local row drift estimations is obtained (together with the maximum correlation scores providing an estimate of their actual similarity).
2. Global offsets are calculated from the local ones. At present the method is very simple as it seems sufficient in most cases: local drift derivatives are fitted for each row onto the local drift estimations and the global drift is then obtained by integration (i.e. summing the local drifts).

Option *Exclude linear skew* subtracts the linear term from the calculated drift, it can be useful when the image is anisotropic and its features are supposed to be oriented in a direction not paralled to the image sides.



Drift correction example: (a) original data exhibiting strong drift in the fast scan axis, (b) corrected data, (c) calculated drift graph.

Affine Distortion

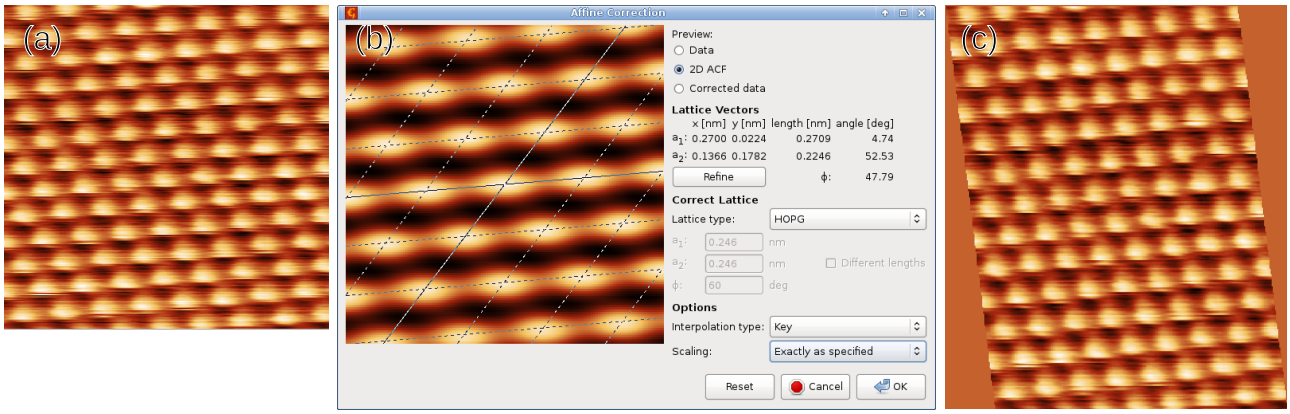
Data Process → Distortion → Affine

Affine distortion in the horizontal plane caused by thermal drift is common for instance in STM. If the image contains a regular structure, for instance an atomic lattice of known parameters, the distortion can be easily corrected using this function.

The affine distortion correction requires to first select the distorted lattice in the image. This is done by moving the lattice selection on the preview with mouse until it matches the regular features present in the image. For images of periodic lattices, it is usually easier to select the lattice in the autocorrelation function image (*2D ACF*). Also, only a rough match needs to be found manually in this case. Button *Refine* refines the selected lattice vectors to the nearest maxima in autocorrelation function with subpixel precision.

If scan lines contain lots of high-frequency noise the horizontal ACF (middle row in the ACF image) can be also quite noisy. In such case it is better to interpolate it from the surrounding rows by enabling the option *Interpolate horizontal ACF*.

The correct lengths of the lattice vectors a_1 and a_2 and the angle φ between them, entered to the dialogue, determine the affine transformation to perform. A few common lattice types (such as HOPG surface) are offered predefined, but it is possible to enter arbitrary lengths and angle.



Affine correction example: (a) original image exhibiting an affine distortion, (b) correction dialogue with the lattice selected on the two-dimensional autocorrelation, (c) corrected image.

It should be noted that the correction method described above causes all lateral scale information in the image to be lost because the new lateral scale is fully determined by the correct lattice vectors. This is usually the best option for STM images of known atomic lattices, however, for a general skew or affine correction it can be impractical. Therefore, the dialogue offers three different scaling choices:

Exactly as specified Lattice vectors in the corrected image will have the specified lengths and angle between them. Scale information of the original image is discarded completely.

Preserve area Lattice vectors in the corrected image will have the specified ratio of lengths and angle between them. However, the overall scale is calculated as to make the affine transformation area-preserving.

Preserve X scale Lattice vectors in the corrected image will have the specified ratio of lengths and angle between them. However, the overall scale is calculated as to make the affine transformation preserve the original x-axis scale. This is somewhat analogous to the scale treatment in Drift compensation.

The function can also correct one image using the ACF calculated from another image (presumably a different channel in the same measurement) or apply the correction to all images in the file. The former is accomplished by selecting a different image as *Image for ACF* in the options. Affine transformation of all images in the file is enabled by *Apply to all compatible images*. Both options requite the other images to be compatible with the current one, i.e. having the same pixel and real dimensions.

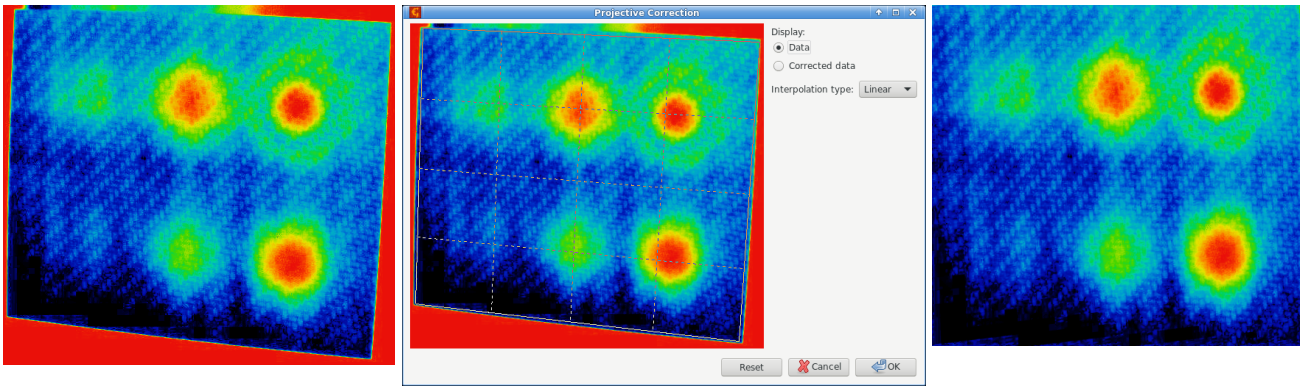
Perspective Distortion

Data Process → *Distortion* → *Perspective*

Perspective distortion is unusual in scanning probe microscopy, but relatively common in other types of image data. The perspective distortion module can correct such distortion if the image contains features which are known to be in fact rectangular.

To apply the correction, a perspective rectangle is selected on the image by moving its four corners. The selected region is then transformed and extracted to the result. The preview can be switched between original and corrected data.

The module sets the pixel and physical dimensions of the result to estimated reasonable values, based on the assumption the distortion is not large. The dimensions then more or less correspond to the dimensions of the selected distorted rectangle in the original image. However, it is just an estimate. If the dimensions of the extracted rectangle can be determined more precisely, for instance using markings in the image, they should be set using Dimensions and Units.



Perspective correction example. Left: original image exhibiting an perspective distortion. Centre: correction dialogue with the rectangle selected in the image. Right: corrected image.

Option *Create new image* controls whether the image is replaced with corrected or a new image created. The function can also apply the correction to all compatible images in the file.

Polynomial Distortion 

Data Process → *Distortion* → *Polynomial*

General distortion in the horizontal plane can be compensated, or created, with Polynomial distortion. It performs transforms that can be expressed as

$$x_{\mathrm{old}} = P_x(x_{\mathrm{new}}, y_{\mathrm{new}}),$$
$$y_{\mathrm{old}} = P_y(x_{\mathrm{new}}, y_{\mathrm{new}}),$$

where P_x and P_y are polynomials up to the third total degree with user-defined coefficients. Note the direction of the coordinate transform – the reverse direction would not guarantee an unambiguous mapping.

The polynomial coefficients are entered as scale-free, i.e. as if the coordinate ranges were always [0, 1]. If *Instant updates* are enabled, pressing **Enter** in a coefficient entry (or just leaving moving keyboard focus elsewhere) updates the preview.

Straighten Path

Data Process → Distortion → Straighten Path

Extraction of data along circular features, curved step edges and other arbitrarily shaped smooth paths can be a useful preprocessing step for further analysis. Function *Straighten Path* creates a new image by "straightening" the input image along a spline path specified using a set of points on the image.

The points defining the path can be defined on the preview image. Each control point can be moved with mouse. Clicking into the empty space adds a new point, connected to the closer end of the path. Individual points can be deleted by selecting them in the point list in the left part of the dialogue and pressing **Delete**.

The shape of the extracted area is controlled by the following parameters:

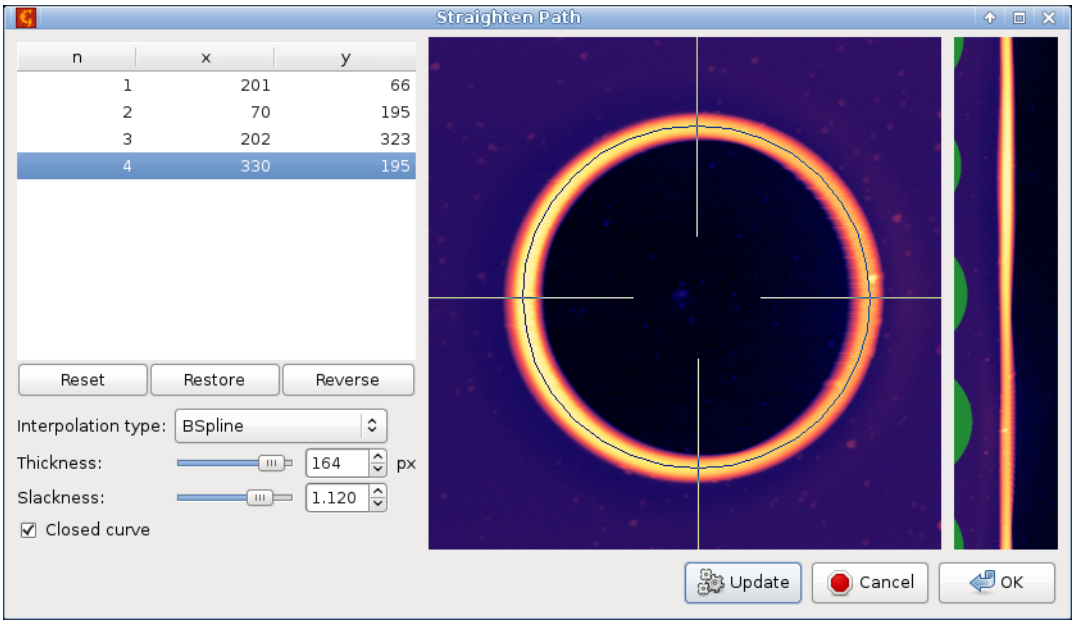
Thickness The width of the extracted image, denoted by the perpendicular marks on the path. This is the full width, i.e. the distance to each side of the path is half of the thickness.

Slackness Parameter influencing the shape of the path between the control points. It can be visualised as the slackness of a string connecting the control points. For zero slackness the string is taut and the path is formed by straight segments. Increasing slackness means decreasing tensile stress in the string, up to slackness of unity for which the string is stress-free. Values larger than unity mean there is an excess string length, leading to compressive stress.

Closed curve If this option is enabled the path is closed, allowing the extraction of ring-shaped areas (for instance). Otherwise the path has two free ends.

Pressing *Update* computes a preview of the extracted image and shows it to the right of the input image. The preview is displayed vertically, with top edge corresponding to the path beginning and bottom edge to its end. The extracted image can be either oriented the same way or horizontally (from left to right), depending on *Output orientation*.

For certain combinations of path shape and thickness the resulting image can contain regions that lie outside the original image. Such regions are masked in the output and filled with a neutral value.



Straighten Path example, showing extraction of data along a closed circular path. The left part of the dialogue contains coordinates of the path-defining points and function options, the middle part shows the selected circular path (with perpendicular markers denoting the thickness), and the right part is a preview of the extracted straightened data.

Extract Path Selection

Data Process → Distortion → Extract Path Selection

Function Straighten Path stores the selected path with the data so that it can be recalled later. If you need the coordinates and/or directions corresponding to the straightened image for further processing you can use *Extract Path Selection*.

The function plots the real coordinates or tangents to the path, i.e. the direction cosines and sines, as graphs. Coordinates are selected as *X position* and *Y position*; directions are selected as *X tangent* and *Y tangent*. The number of points of each graph curve is the same as the number of rows of the straightened image created by Straighten Path and there is one-to-one correspondence between the rows and graph curve points.

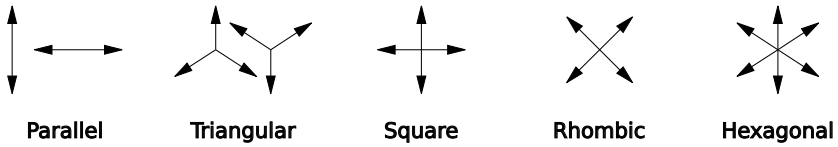
Unrotate

Data Process → *Correct Data* → *Unrotate*

Unrotate can automatically make principal directions in an image parallel with horizontal and/or vertical image edges. For that to work, the data need to have some principal directions, therefore it is most useful for scans of artificial and possibly crystalline structures.

The rotation necessary to straighten the image – displayed as *Correction* – is calculated from peaks in angular slope distribution assuming a prevalent type of structure, or symmetry. The symmetry can be estimated automatically too, but it is possible to select a particular symmetry type manually and let the module calculate only corresponding rotation correction. Note if you assume a structure type that does not match the actual structure, the calculated rotation is rarely meaningful.

It is recommended to level (or facet-level) the data first as overall slope can skew the calculated rotations.



Orientations of prevalent directions corresponding to Unrotate symmetry types.

The assumed structure type can be set with *Assume* selector. Following choices are possible:

Detected Automatically detected symmetry type, displayed above as *Detected*.

Parallel Parallel lines, one prevalent direction.

Triangular Triangular symmetry, three prevalent directions (unilateral) by 120 degrees.

Square Square symmetry, two prevalent directions oriented approximately along image sides.

Rhombic Rhombic symmetry, two prevalent directions oriented approximately along diagonals. The only difference from Square is the preferred diagonal orientation (as opposed to parallel with sides).

Hexagonal Hexagonal symmetry, three prevalent directions (bilateral) by 120 degrees.

Periodic Translation

Data Process → *Correct Data* → *Translate Periodically*

In periodic images, it can be useful to move the origin of the *xy* plane as it does not create any discontinuities. The same is true for images which may not be strictly periodic but, for instance, contain one dominant feature and are almost zero elsewhere.

The function moves a selected point in the image either to the centre or to the top-left corner, as specified with *Move selected point to*. The point coordinates can be entered numerically as *Translation* or by clicking in the preview image. Since the point is selected in the original image the second method is only possible when the original image is displayed (controlled by *Display*).

If the checkbox *Update coordinate offsets* is enabled the function adjusts the coordinate origin to preserve coordinates. Meaning an image feature will appear at the same coordinates after the transformation. Periodicity leads to some ambiguity here. In fact, there are infinite equally valid choices for the origin. The origin is chosen to avoid too large coordinates. When the checkbox is unchecked the lateral coordinates are kept unchanged.

Displacement Field

Displacement field is a general method of image distortion in the *xy* plane. A vector is assigned to each point in the output image. This vector determines how far and in which direction to move from this point. The value found at the shifted position in the source image becomes the value of the pixel in the output image:

$$z(x,y) = z_{\text{source}}\big(x + v_x(x,y), y + v_y(x,y)\big)$$

where **v** is the displacement vector.

The displacement field can be generated or obtained by several methods:

Gaussian (scan lines) The vectors have only the x component. The displacement is one-dimensional, along scan lines (image rows). It is a random Gaussian image, similar to Gaussian images generated by Spectral Synthesis.

Gaussian (two-dimensional) The vectors have both x and y components, so the displacement is two-dimensional. It is again a random Gaussian image.

Tear scan lines The vectors have only the x component. The displacement is one-dimensional, along scan lines (image rows). It is generated to tear segments of neighbour rows by shifting one to the left and another to the right. In the rest of the image the displacement is smooth (and close to zero) so continuity is preserved outside the tear.

Image (scan lines) The vectors have only the x component. The displacement is one-dimensional, along scan lines (image rows), and given by another image (selected as *X displacement*). The values of the image are directly used as the displacement vectors and must have the same units as lateral coordinates of the deformed image.

Image (two-dimensional) The vectors have both x and y components, so the displacement is two-dimensional. They are given by another images (selected as *X displacement* and *Y displacement*). The values of the images are directly used as the displacement vectors and must have the same units as lateral coordinates of the deformed image.

Coerce 

Data Process → *Distortion* → *Coerce*

The module transforms the values in a height field to enforce certain prescribed statistical properties. The distribution of values can be transformed to Gaussian, with mean value and rms roughness preserved, or uniform, with minimum and maximum value preserved. It is also possible to enforce the same distribution as another height field selected as *Template*.

The transformation preserves the ordering of the values, i.e. if one point is higher than another before the transformation it will be also higher after the transformation. Option *Data processing* controls whether the transformation is applied to the image as a whole (*Entire image*) or each row separately, with the same distribution (*By row (identically)*). Note that in the latter case if a template is selected each row of the transformed image will have the same value distribution (approximately) as the template.

In addition, a different kind of transformation can be performed, discretisation of the values into given number of levels. This operation is selected with *Discrete levels*. There are two different discretisations available, either the height levels are equidistant from minimum to maximum value (*Uniform*) or they are calculated so that after the transformation each level will consist of approximately the same number of pixels (*Same area*).

Radial Smoothing

Data Process → *Distortion* → *Radial Smoothing*

Radial smoothing processes the image with a Gaussian smoothing filter, but in polar coordinates. This function is useful for images which should be relatively smooth and exhibit radial symmetry.

The image can be smoothed either along radial profiles – with Gaussian filter width controlled by *Radius* – or angularly along arcs of constant radius from the centre – with width controlled by *Angle*. The transformation to polar coordinates and back does not preserve the image perfectly. Subtle (and occasionally less subtle) artefacts can occur, in particular close to the centre, even with no actual smoothing.

4.10 Statistical Analysis

While analysing randomly rough surfaces we often need a statistical approach to determine some set of representative quantities. Within Gwyddion, there are several ways of doing this. In this section we will explain the various statistical tools and modules offered in Gwyddion, and also present the basic equations which were used to develop the algorithms they utilize.

Scanning probe microscopy data are usually represented as a two-dimensional data field of size $N \times M$, where N and M are the number of rows and columns of the data field, respectively. The real area of the field is denoted as $L_x \times L_y$ where L_x and L_y are the dimensions along the respective axes. The sampling interval (distance between two adjacent points within the scan) is denoted Δ. We assume that the sampling interval is the same in both the x and y direction and that the surface height at a given point (x, y) can be described by a random function $\xi(x, y)$ that has given statistical properties.

Note that the AFM data are usually collected as line scans along the x axis that are concatenated together to form the two-dimensional image. Therefore, the scanning speed in the x direction is considerably higher than the scanning speed in the y direction. As a result, the statistical properties of AFM data are usually collected along the x profiles as these are less affected by low frequency noise and thermal drift of the sample.

Statistical Quantities Tool

Statistical quantities include basic properties of the height values distribution, such as variance, skewness and kurtosis. The quantities accessible within Gwyddion by means of the Statistical Quantities tool are divided into several groups.

Moment-Based

Moment based quantities are expressed using integrals of the height distribution function with some powers of height. They include the familiar quantities:

- Mean value.
- Mean square roughness or RMS of height irregularities S_q: this quantity is computed from 2nd central moment of data values.
- Grain-wise RMS value which differs from the ordinary RMS only if masking is used. The mean value is then determined for each grain (contiguous part of mask or inverted mask, depending on masking type) separately and the variance is then calculated from these per-grain mean values.
- Mean roughness or S_a value of height irregularities.
- Height distribution skewness computed from 3rd central moment of data values.
- Height distribution kurtosis computed from 4th central moment of data values.

More precisely, RMS (σ), skewness (γ_1), and kurtosis (γ_2) are computed from central moments of i-th order μ_i

$$\mu_i = \frac{1}{N}\sum_{n=1}^{N}(z_n - \bar{z})^i$$

where $\bar{z}$ denotes the mean value. The are expressed by the following formulas:

$$\sigma = \mu_2^{1/2}, \quad \gamma_1 = \frac{\mu_3}{\mu_2^{3/2}}, \quad \gamma_2 = \frac{\mu_4}{\mu_2^2} - 3$$

Note that Gwyddion computes the *excess* kurtosis, which is zero for Gaussian data distribution. Add 3 to obtain the area texture parameter Sku.

Mean roughness S_a is similar to RMS value, the difference being that it is calculated from the sum of absolute values of data differences from the mean, instead of their squares.

It should be noted that quantities such as σ calculated using the standard definitions are biased, especially if any scan line levelling has been done. The Correlation Length tool can calculate bias-corrected quantities. See its description for more details.

Order-Based

Order based quantities represent values corresponding to specific ranks if all the values were ordered. They include:

- The minimum and maximum value and median.
- Maximum peak height S_p, which differs from the maximum by being calculated with respect to the mean height.
- Maximum pit depth S_v, which differs from the minimum by being calculated with respect to the mean height.
- Maximum height S_z, the total value range. It is the difference between minimum and maximum or, equivalently, S_v and S_p.

Hybrid

Hybrid quantities combine heights and spatial relations in the surface and include:

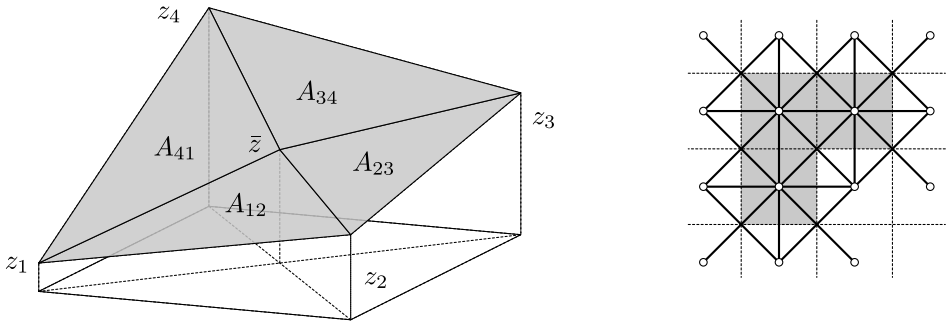
- Projected surface area and surface area: computed by simple triangulation.
- Volume, calculated as the integral of the surface height over the covered area.
- Mean inclination of facets in area: computed by averaging normalized facet direction vectors.
- Variation, which is calculated as the integral of the absolute value of the local gradient.
- Surface slope, which is the mean square local gradient. In other words, the mean square of the local gradient is first calculated. The slope is then obtained as its square root.

The surface area is estimated by the following method. Let z_i for $i = 1,2,3,4$ denote values in four neighbour points (pixel centres), and h_x and h_y pixel dimensions along corresponding axes. If an additional point is placed in the centre of the rectangle which corresponds to the common corner of the four pixels (using the mean value of the pixels), four triangles are formed and

the surface area can be approximated by summing their areas. This leads to the following formulas for the area of one triangle (top) and the surface area of one pixel (bottom):

$$A_{12} = \frac{h_x h_y}{4}\sqrt{1+\left(\frac{z_1-z_2}{h_x}\right)^2+\left(\frac{z_1+z_2-2\bar{z}}{h_y}\right)^2}$$
$$A = A_{12}+A_{23}+A_{34}+A_{41}$$

The method is now well-defined for inner pixels of the region. Each value participates on eight triangles, two with each of the four neighbour values. Half of each of these triangles lies in one pixel, the other half in the other pixel. By counting in the area that lies inside each pixel, the total area is defined also for grains and masked areas. It remains to define it for boundary pixels of the whole data field. We do this by virtually extending the data field with a copy of the border row of pixels on each side for the purpose of surface area calculation, thus making all pixels of interest inner.



Surface area calculation triangulation scheme (left). Application of the triangulation scheme to a three-pixel masked area (right), e.g. a grain. The small circles represent pixel-centre vertices z_i, thin dashed lines stand for pixel boundaries while thick lines symbolize the triangulation. The surface area estimate equals to the area covered by the mask (grey) in this scheme.

Other

In addition, the tool calculates a few quantities that do not belong in any of the previous categories:

- Scan line discrepancy, characterising the differences between scan lines. It can be visualised as follows: if each line in the image was replaced by the average of the two neighbour lines, the new image would differ slightly from the original. Taking the mean square difference and dividing it by the mean square value of the image, we obtain the displayed discrepancy value.

Tip By default, the Statistical Quantities tool will display figures based on the entire image. If you would like to analyse a certain region within the image, simply click and drag a rectangle around it. The tool window will update with new numbers based on this new region. If you want you see the statistical quantities for the entire image again, just click once within the data window and the tool will reset.

Statistical Functions Tool

One-dimensional statistical functions can be accessed by using the Statistical Functions tool. Within the tool window, you can select which function to evaluate using the selection box on the left labelled *Output Type*. The graph preview will update automatically. You can select in which direction to evaluate (horizontal or vertical), but as stated above, we recommend using the fast scanning axis direction. You can also select which interpolation method to use. When you are finished, click *Apply* to close the tool window and output a new graph window containing the statistical data.

Tip Similar to the Statistical Quantities tool, this tool evaluates for the entire image by default, but you can select a sub-region to analyse if you wish.

Height and Angle Distribution Functions

The simplest statistical functions are the height and slope distribution functions. These can be computed as non-cumulative (i.e. densities) or cumulative. These functions are computed as normalized histograms of the height or slope (obtained as derivatives in the selected direction – horizontal or vertical) values. In other words, the quantity on the abscissa in "angle distribution" is the tangent of the angle, not the angle itself.

The normalization of the densities $\rho(p)$ (where p is the corresponding quantity, height or slope) is such that

$$\int_{-\infty}^{\infty} \rho(p)\,\mathrm{d}p = 1$$

Evidently, the scale of the values is then independent on the number of data points and the number of histogram buckets. The cumulative distributions are integrals of the densities and they have values from interval $[0,1]$.

First-Order vs. Second-Order Quantities

The height and slope distribution quantities belong to the first-order statistical quantities, describing only the statistical properties of the individual points. However, for the complete description of the surface properties it is necessary to study higher order functions. Usually, second-order statistical quantities observing mutual relationship of two points on the surface are employed. These functions are namely the autocorrelation function, the height-height correlation function, and the power spectral density function. A description of each of these follows:

Autocorrelation Function

The autocorrelation function is given by

$$\begin{aligned} G(\tau_x,\tau_y) &= \iint_{-\infty}^{\infty} z_1 z_2 w(z_1,z_2,\tau_x,\tau_y)\,\mathrm{d}z_1\,\mathrm{d}z_2 \\ &= \lim_{S\to\infty} \frac{1}{S} \iint_S \xi(x_1,y_1)\,\xi(x_1+\tau_x,y_1+\tau_y)\,\mathrm{d}x_1\,\mathrm{d}y_1 \end{aligned}$$

where z_1 and z_2 are the values of heights at points (x_1,y_1), (x_2,y_2); furthermore, $\tau_x = x_1 - x_2$ and $\tau_y = y_1 - y_2$. The function $w(z_1,z_2,\tau_x,\tau_y)$ denotes the two-dimensional probability density of the random function $\xi(x,y)$ corresponding to points (x_1,y_1), (x_2,y_2), and the distance between these points τ.

From the discrete AFM data one can evaluate this function as

$$G(m,n) = \frac{1}{(N-n)(M-m)} \sum_{l=1}^{N-n} \sum_{k=1}^{M-m} z_{k+m,l+n} z_{k,l}$$

where $m = \tau_x/\Delta x$, $n = \tau_y/\Delta y$. The function can thus be evaluated in a discrete set of values of τ_x and τ_y separated by the sampling intervals Δx and Δy, respectively.

For AFM measurements, we usually evaluate the one-dimensional autocorrelation function based only on profiles along the fast scanning axis. It can therefore be evaluated from the discrete AFM data values as

$$G_x(m) = G(m,0) = \frac{1}{N(M-m)} \sum_{l=1}^{N} \sum_{k=1}^{M-m} z_{k+m,l} z_{k,l}$$

It should be noted that this estimate is biased, especially if any scan line levelling has been done. The Correlation Length too can calculate a bias-corrected ACF. See its description for more details.

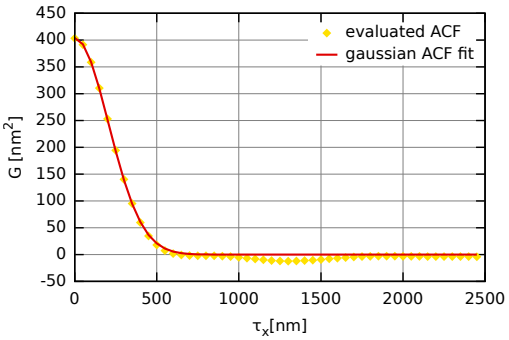
The one-dimensional autocorrelation function is often assumed to have the form of a Gaussian, i.e. it can be given by the following relation

$$G_x(\tau_x) = \sigma^2 \exp(-\tau_x^2/T^2)$$

where σ denotes the root mean square deviation of the heights and T denotes the autocorrelation length.

For the exponential autocorrelation function we have the following relation

$$G_x(\tau_x) = \sigma^2 \exp(-\tau_x/T)$$



Autocorrelation function obtained for simulated Gaussian randomly rough surface (i.e. with a Gaussian autocorrelation function) with $\sigma \approx 20\,\mathrm{nm}$ and $T \approx 300\,\mathrm{nm}$.

We can also introduce the radial ACF $G_\mathrm{r}(\tau)$, i.e. angularly averaged two-dimensional ACF, which of course contains the same information as the one-dimensional ACF for isotropic surfaces:

$$G_\mathrm{r}(\tau) = \int_0^{2\pi} W(\tau\cos\varphi, \tau\sin\varphi)\,\mathrm{d}\varphi$$

Note For optical measurements (e.g. spectroscopic reflectometry, ellipsometry) the Gaussian autocorrelation function is usually expected to be in good agreement with the surface properties. However, some articles related with surface growth and oxidation usually assume that the exponential form is closer to the reality.

Interface Distribution Function

The interface distribution function (IDF) is the second derivative of ACF

$$I(\tau) = \frac{\mathrm{d}^2}{\mathrm{d}\tau^2} G(\tau)$$

Positions and signs of its extrema are linked to features like grains sizes and intergrain distances.

Height-Height Correlation Function

The difference between the height-height correlation function and the autocorrelation function is very small. As with the autocorrelation function, we sum the multiplication of two different values. For the autocorrelation function, these values represented the different distances between points. For the height-height correlation function, we instead use the power of difference between the points.

For AFM measurements, we usually evaluate the one-dimensional height-height correlation function based only on profiles along the fast scanning axis. It can therefore be evaluated from the discrete AFM data values as

$$H_x(\tau_x) = \frac{1}{N(M-m)} \sum_{l=1}^{N} \sum_{n=1}^{M-m} (z_{n+m,l} - z_{n,l})^2$$

where $m = \tau_x/\Delta x$. The function thus can be evaluated in a discrete set of values of τ_x separated by the sampling interval Δx.

The one-dimensional height-height correlation function is often assumed to be Gaussian, i.e. given by the following relation

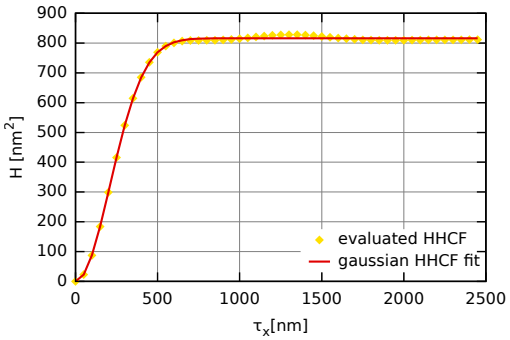
$$H_x(\tau_x) = 2\sigma^2 \left[1 - \exp\left(-\frac{\tau_x^2}{T^2}\right)\right]$$

where σ denotes the root mean square deviation of the heights and T denotes the autocorrelation length.

For the exponential height-height correlation function we have the following relation

$$H_x(\tau_x) = 2\sigma^2 \left[1 - \exp\left(-\frac{\tau_x}{T}\right)\right]$$

In the following figure the height-height correlation function obtained for a simulated Gaussian surface is plotted. It is fitted using the formula shown above. The resulting values of σ and T obtained by fitting the HHCF are practically the same as for the ACF.



Height-height correlation function obtained for simulated Gaussian randomly rough surface with $\sigma \approx 20\,\mathrm{nm}$ and $T \approx 300\,\mathrm{nm}$.

Power Spectral Density Function

The two-dimensional power spectral density function can be written in terms of the Fourier transform of the autocorrelation function

$$W(K_x, K_y) = \frac{1}{4\pi} \iint_{-\infty}^{\infty} G(\tau_x, \tau_y)\, \mathrm{e}^{-\mathrm{i}(K_x\tau_x + K_y\tau_y)}\, \mathrm{d}\tau_x\, \mathrm{d}\tau_y$$

Similarly to the autocorrelation function, we also usually evaluate the one-dimensional power spectral density function which is given by the equation

$$W_1(K_x) = \int_{-\infty}^{\infty} W(K_x, K_y)\, \mathrm{d}K_y$$

This function can be evaluated by means of the Fast Fourier Transform as follows:

$$W_1(K_x) = \frac{2\pi}{NMh} \sum_{j=0}^{M-1} |\hat{P}_j(K_x)|^2$$

where $P_j(K_x)$ is the Fourier coefficient of the j-th row, i.e.

$$\hat{P}_j(K_x) = \frac{h}{2\pi} \sum_{n=0}^{N-1} z_{nj} \exp(-\mathrm{i}K_x n h)$$

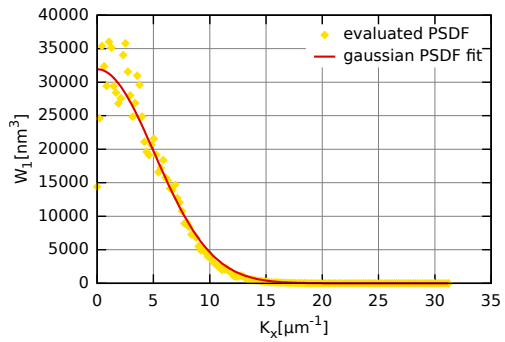
If we choose the Gaussian ACF, the corresponding Gaussian relation for the PSDF is

$$W_1(K_x) = \frac{\sigma^2 T}{2\sqrt{\pi}} \exp(-K_x^2 T^2/4)$$

For the surface with exponential ACF we have

$$W_1(K_x) = \frac{\sigma^2 T}{\pi} \frac{1}{1 + K_x^2 T^2}$$

In the following figure the resulting PSDF and its fit for the same surface as used in the ACF and HHCF fitting are plotted. We can see that the function can be again fitted by Gaussian PSDF. The resulting values of σ and T were practically same as those from the HHCF and ACF fit.



PSDF obtained for data simulated with Gaussian autocorrelation function.

We can also introduce the radial PSDF $W_r(K)$, i.e. angularly integrated two-dimensional PSDF, which of course contains the same information as the one-dimensional PSDF for isotropic rough surfaces:

$$W_r(K) = \int_0^{2\pi} W(K\cos\varphi, K\sin\varphi)\, K\, d\varphi$$

For a surface with Gaussian ACF this function is expressed as

$$W_r(K) = \frac{\sigma^2 T}{2} KT \exp(-K^2T^2/4)$$

while for exponential-ACF surface as

$$W_r(K) = \sigma^2 T \frac{KT}{(1+K^2T^2)^{3/2}}$$

Tip Within Gwyddion you can fit all statistical functions presented here by their Gaussian and exponential forms. To do this, fist click *Apply* within the Statistical Functions tool window. This will create a new graph window. With this new window selected, click on *Graph* → *Fit Graph*.

The spectral density can also be integrated radially, giving the angular spectrum. Its peaks can be used to identify the image texture direction.

Minkowski Functionals

The Minkowski functionals are used to describe global geometric characteristics of structures. Two-dimensional discrete variants of volume V, surface S and connectivity (Euler-Poincaré Characteristic) χ are calculated according to the following formulas:

$$V = \frac{N_{\text{white}}}{N}, \quad S = \frac{N_{\text{bound}}}{N}, \quad \chi = \frac{C_{\text{white}} - C_{\text{black}}}{N}$$

Here N denotes the total number of pixels, N_{white} denotes the number of "white" pixels, that is pixels above the threshold. Pixels below the threshold are referred to as "black". Symbol N_{bound} denotes the number of white-black pixel boundaries. Finally, C_{white} and C_{black} denote the number of continuous sets of white and black pixels respectively.

For an image with continuous set of values the functionals are parametrized by the height threshold value ϑ that divides white pixels from black, that is they can be viewed as functions of this parameter. And these functions $V(\vartheta)$, $S(\vartheta)$ and $\chi(\vartheta)$ are plotted.

Range

The range distribution is a plot of the growth of value range depending on the lateral distance. It is always a non-decreasing function.

For each pixel in the image and each lateral distance, it is possible to calculate the minimum and maximum from values that do not lie father than the given lateral distance from the given pixel. The local range is the difference between the maximum and minimum (and in two dimensions can be visualised using the Rank presentation function). Averaging the local ranges for all image pixels gives the range curve.

Area Scale Graph

The area scale graph shows the ratio of developed surface area and projected area minus one, as the function of scale at which the surface area is measured. Because of the subtraction of unity, which would be the ratio for a completely flat surface, the quantity is called "excess" area.

Similar to the correlation functions, the area excess can be in principle defined as a directional quantity. However, the function displayed in the tool assumes an isotropic surface – the horizontal or vertical orientation that can be chosen there only determines the primary direction used in its calculation.

Masked Statistical Functions

Most of the statistical functions support masking. In other words, they can be calculated for arbitrarily shaped image regions. For some of them, such as the height or angle distribution, no further explanation is needed: only image pixels covered (or not covered) by the mask are included in the computation. However, the meaning of the function is less clear for correlation functions and spectral densities.

We will illustrate it for the ACF (the simplest case). The full description can be found in the literature [1]. The discrete ACF formula can be rewritten

$$G_x(m) = G(m,0) = \frac{1}{|\Omega_m|} \sum_{(k,l)\in\Omega_m} z_{k+m,l} z_{k,l}$$

where Ω_m is the set of image pixels inside the image region for which the pixel m columns to the right also lies inside the region. If we calculate the ACF for a rectangular region, for example the entire image, nothing changes. However, this formula expresses the function meaningfully for regions of any shape. The only missing part is how to perform the computation efficiently.

For this we define $c_{k,l}$ as the mask of valid pixels, i.e. it is equal to 1 for pixels we want to include and 0 for pixels we want to exclude. It then follows that

$$|\Omega_m| = \sum_{l=1}^{N} \sum_{k=1}^{M-m} c_{k+m,l} c_{k,l}$$

Furthermore, if image data are premultiplied by $c_{k,l}$ the ACF sum over the irregular region can be replaced by the usual ACF sum over the entire image. Both are efficiently computed using FFT.

It is important to note that the amount of information available in the data about the ACF for given m depends on Ω_m. Unlike for entire rectangular images, it does not have to decrease monotonically with increasing m and can vary almost arbitrarily. There can even be holes, i.e. horizontal distances m for which the region contains no pair of pixels exactly this apart. If this occurs Gwyddion replaces the missing values using linear interpolation.

The height-height correlation function is calculated in a similar manner, only the sums have to be split into parts which can be then evaluated using FFT. For PSDF, this is not possible because each Fourier coefficient depends on each data value. So, instead, the cyclic autocorrelation function is calculated in the same manner as ACF, and PSDF is then obtained as its Fourier transform.

Row/Column Statistics Tool 

The Row/Column Statistics tool calculates numeric characteristics of each row or column and plots them as a function of its position. This makes it kind of complementary to Statistical Functions tool. Available quantities include:

- Mean value, minimum, maximum and median.
- RMS value of the height irregularities computed from data variance R_q.
- Skewness and kurtosis of the height distribution.
- Surface line length. It is estimated as the total length of the straight segments joining data values in the row (column).
- Overall slope, i.e. the tangent of the mean line fitted through the row (column).
- Tangent of β_0. This is a characteristics of the steepness of local slopes, closely related to the behaviour of autocorrelation and height-height correlation functions at zero. For discrete values it is calculated as follows:

$$\tan^2 \beta_0 = \frac{1}{(N-1)h^2} \sum_{i=1}^{N-1} (z_i - z_{i-1})^2$$

- Standard roughness parameters Ra, Rz, Rt.

In addition to the graph displaying the values for individual rows/columns, the mean value and standard deviation of the selected quantity is calculated from the set of individual row/column values. It must be emphasised that the standard deviation of the selected quantity represents the dispersion of values for individual rows/columns and cannot be interpreted as an error of the analogous two-dimensional quantity.

Correlation Length Tool 

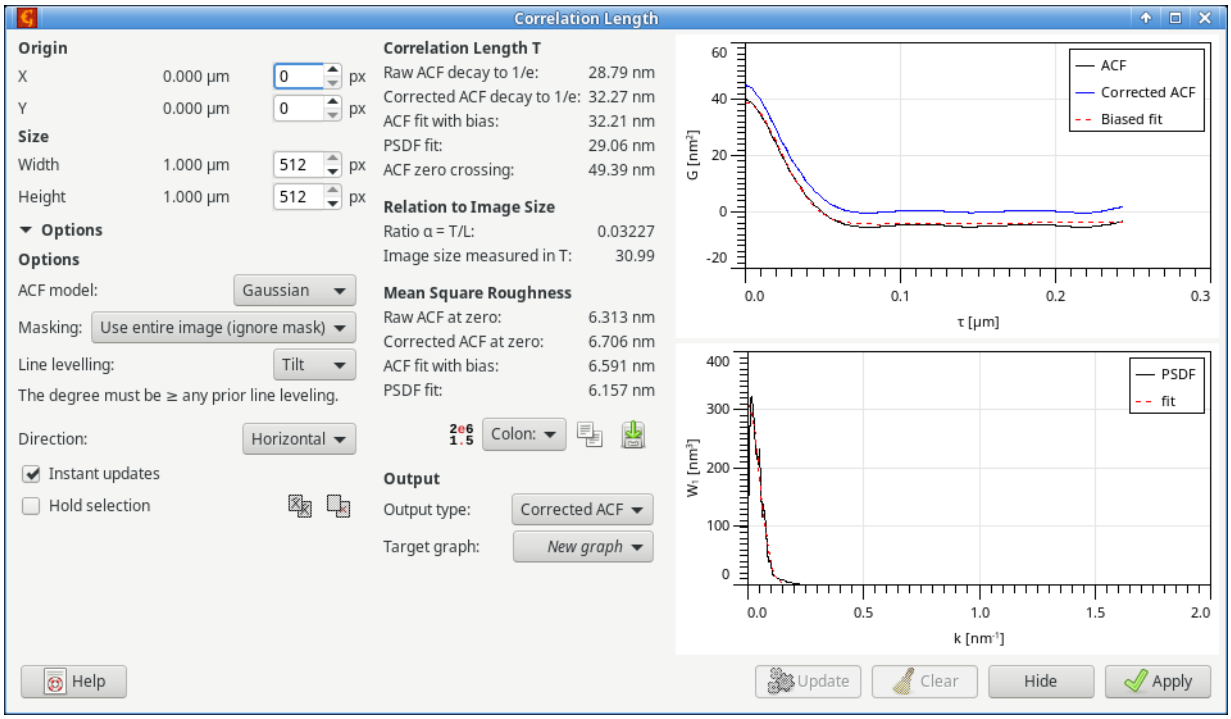
The Correlation Length tool can be used for a one-step estimation of image autocorrelation length T from scan lines. The autocorrelation length is estimated using several common methods using either the 1D autocorrelation or 1D power spectral density function. Both functions are plotted on the right side of the dialogue togther with additional information such as their fits by models.

On the left side one can choose the evaluation parameters (beside the common *Masking* and *Orientation* options):

ACF model Autocorrelation function type assumed for fitting ACF and PSDF with analytcal models. The model is selected for ACF and the tool uses matching PSDF model automatically.

Line levelling The tool can performs scan line levelling during the estimation (in the selected evaluation direction, which is usually horizontal). It is possible to not apply any levelling by selecting *None* if lines have already been levelled for instance using Align Rows. However, the bias-corrected evaluation needs to know how they were levelled. You must select *degree at least as high as any prior line levelling*. Otherwise the results are not valid.

The results of all evaluations are displayed in the middle column, allowing the user to choose the most appropriate one (and compare immediately how much they differ). Note, however, that the `raw' results are biased [2,3] and mainly for comparison and in general should not be used. For ACF-based evaluation, the bias-corrected values [4] are preferred.



Correlation length tool with typical results for a rough surface.

Reported values are split into several section. The correlation length related are defined:

Raw ACF decay to 1/e The horizontal distance at which the discrete ACF (curve *ACF*) decays to $1/e$ of its maximum value, which it attains at zero. This is the common definition, matching also the expressions for Gaussian and exponential ACF.

Corrected ACF decay to 1/e The distance at which the bias-corrected ACF (curve *Corrected ACF*) decays to $1/e$ of its maximum value.

ACF fit with bias Correlation length obtained from fitting the raw ACF with the selected model. A modified model is used which takes into account that the discrete ACF data are biased [4]. The fit is shown as curve *Biased fit*. Note that biased should be read as bias-corrected in this context.

PSDF fit Autocorrelation length obtained by fitting the estimated PSDF using the selected model. The model is not modified in any manner, even though a few initial PSDF data points (the most affected by levelling) are excluded.

ACF zero crossing The horizontal distance at which the raw ACF decays to zero. An uncommon, but sometimes useful characteristics of the ACF decay. The zero crossing is also used in the bias-corrected evaluation.

Image size related quantities compare autocorrelation length T and image size L (more precisely scan line length):

Ratio $\alpha = T/L$ This ratio is used for the estimation of bias of roughness parameters due to finite measurement area. The autocorrelation length is taken from the extrapolated decay.

Image size measured in T The inverse of α measures how many autocorrelation lengths fit into one scan line. It carries the same information, but can be easier to imagine.

A crude estimate of the bias can be obtained as follows. It is mainly useful for getting an idea whether the scan size is sufficient, or how much the result may biased – attempting to use it for correction can easily make things worse.

1. Estimate α – for instance by this tool.
2. Multiply it by the degree of polynomial used for scan line levelling plus one. This means 1 for subtraction of mean value, 2 for tilt correction, 3 for bow correction, etc.
3. The result estimates *relative* bias of measued roughness. The bias is always negative so add minus if it matters.

A similar procedure can be used for 2D background subtraction. However, almost always some row levelling is applied – at least implicitly the subtraction of mean value. And it is then also the dominant source of bias.

Finally, mean square roughness values are also reported:

Raw ACF at zero Mean square roughness calculated from the raw ACF using the relation

$$\sigma^2 = G(0)$$

It corresponds to the standard definition.

Corrected ACF at zero Mean square roughness calculated using the same relation, but from the corrected ACF.

ACF fit with bias Mean square roughness obtained from fitting the raw ACF with the selected model. A modified model is used which takes into account that the discrete ACF data are biased. The fit is shown as curve *Biased fit*. Note that biased should be read as bias-corrected in this context.

PSDF fit Mean square roughness obtained by fitting the estimated PSDF using the selected model. The model is not modified in any manner, even though a few initial PSDF data points (the most affected by levelling) are excluded.

The ACF bias correction is described in detail in literature [4]. However, the basic principle is simple. It can be shown that ACF computed from measured data is (in expectation)

$$G_{\mathrm{meas}}(\tau) = (1-R)G_{\mathrm{true}}(\tau)$$

where R is a rather complicated linear operator expressing the bias. It depends on the profile length and scan line levelling applied prior to ACF evaluation. The relation can be inverted to obtain bias-corrected experimental ACF. When ACF is fitted with a known model, an even better approach exists. The model can be modified using $1-R$ to match the experimental ACF curve. This is how the values labelled *Biased fit* are obtained.

Two-Dimensional Autocorrelation Function

Data Process → *Statistics* → *2D Autocorrelation*

The full two-dimensional autocorrelation function introduced above is sometimes used to evaluate anisotropy of rough surfaces. The 2D ACF module calculates the function and can also plot its section in selected directions or calculate several roughness parameters that are based on 2D ACF.

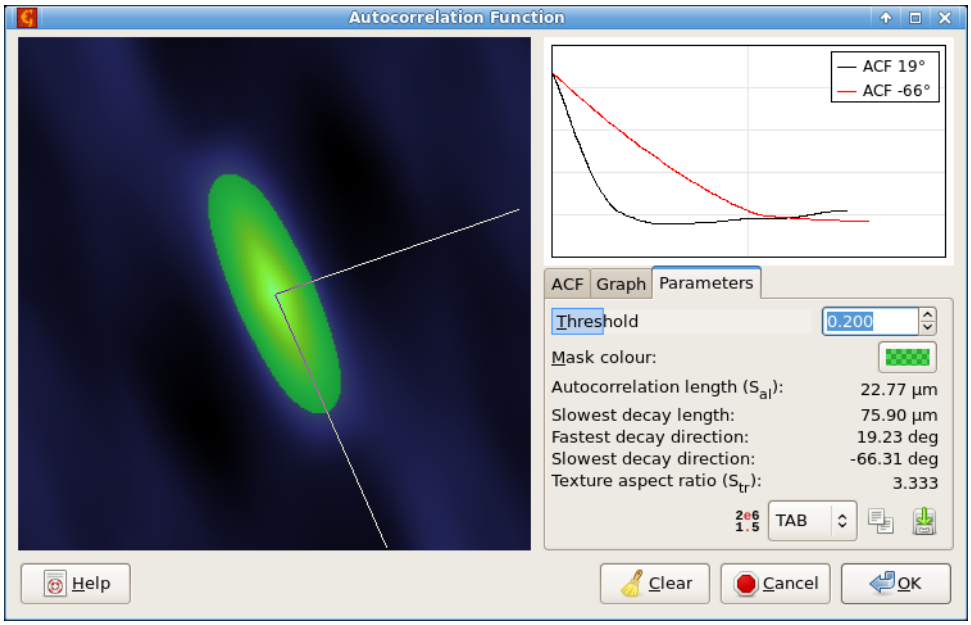
Options in the *ACF* tab control how the function is calculated and displayed. In most cases the mean value should be subtracted before the calculation (as would Mean Zero Value do). This is selected by *Mean value subtraction* in *Data adjustment*. However, *Plane levelling* can be also useful. If you already adjusted the zero level correctly, select *None* to calculate the ACF from unaltered data. If there is a mask on the image the dialogue offers the standard masking options. See Masked Statistics Functions for an overview of masked ACF calculation.

The *Graph* tab contains settings for extracted ACF sections. They are the same as for normal profile extraction.

Finally, *Parameters* displays various numerical parameters derived from 2D ACF. They are determined from the decay of ACF in different directions and depend on *Threshold*, below which data points are considered essentially uncorrelated. The area above the selected threshold is marked using a mask on the ACF image.

The threshold is a fraction with respect to the maximum (occurring at the origin). In theoretical models the autocorrelation length corresponds to decay of ACF to $1/e$. This is also the default value. However, in roughness standards it is usually chosen as 0.2. For comparable results, check that you use consistently one threshold value.

The autocorrelation length S_{al} is the shortest distance at which the ACF falls below the threshold. The corresponding direction is also shown, as well as the distance and direction of the slowest decay. The ration between the longest and smallest distances is the texture aspect ratio S_{tr}.



Two-dimensional ACF dialogue screenshot showing the table with calculated roughness parameters.

Two-Dimensional Spectral Density

Data Process → *Statistics* → *2D PSDF*

The full two-dimensional spectral density function is used to evaluate characteristic spatial frequencies occurring in surfaces. The 2D PSDF module calculates the function and can also plot its section in selected directions.

Options in the *PSDF* tab control how the function is calculated and displayed. Option *Windowing* selects the windowing type. If there is a mask on the image the dialogue offers the standard masking options. See Masked Statistics Functions for an overview of masked PSDF calculation.

The *Graph* tab contains settings for extracted PSDF sections. They are the same as for normal profile extraction.

Finally, *Parameters* displays some numerical parameters derived from 2D PSDF. Two are calculated using the angular spectrum, texture direction S_{td} and texture direction index S_{tdi}. S_{td} is given by the direction in which the radial PSDF integral is the largest and determines the dominant texture direction. S_{tdi} measures how dominant is the dominant direction by comparing the average over all directions to the dominant direction. Note that smaller S_{tdi} means more spectral weight in the dominant direction (not less).

Two-Dimensional Slope Statistics

Several functions in *Data Process* → *Statistics* operate on two-dimensional slope (derivative) statistics.

Slope Distribution calculates a plain two-dimensional distribution of derivatives, that is the horizontal and vertical coordinate on the resulting data field is the horizontal and vertical derivative, respectively. The slopes can be calculated as central derivatives (one-side on the borders of the image) or, if *Use local plane fitting* is enabled, by fitting a local plane through the neighbourhood of each point and using its gradient.

Slope Distribution can also plot summary graphs representing one-dimensional distributions of quantities related to the local slopes and facet inclination angles given by the following formula:

$$\mathbf{v} = \left(\frac{\mathrm{d}z}{\mathrm{d}x}, \frac{\mathrm{d}z}{\mathrm{d}y}\right), \quad v = |\mathbf{v}|, \quad \vartheta = \operatorname{atan} v, \quad \varphi = \operatorname{atan}_2(v_y, -v_x)$$

Three different plots are available:

- *Inclination (ϑ)*, the distribution of the inclination angle ϑ from the horizontal plane. Of course, representing the slope as an angle requires the value and the dimensions to be the same physical quantity.
- *Inclination (gradient)* is similar to the ϑ plot, except the distribution of the derivative v is plotted instead of the angle.
- *Inclination (φ)* visualises the integral of v^2 for each direction φ in the horizontal plane. This means it is not a plain distribution of φ because areas with larger slopes contribute more significantly than flat areas.

Angle Distribution function is a visualization aid that does not calculate a distribution in the strict sense. For each derivative $\mathbf{v}$ the circle of points satisfying

$$2\mathbf{r}\cdot\mathbf{v} = r^2$$

is drawn. The number of points on the circle is given by *Number of steps*.

Entropy

Data Process → *Statistics* → *Entropy*

This function can estimate the differential entropy of value and slope distributions and also provides a visualisation of how it is calculated.

The Shannon differential entropy for a probability density function can be expressed

$$S = -\int_X p(x)\log p(x)\,\mathrm{d}x$$

where X is the domain of the variable x. For instance for the height distribution x represents the surface height and X is the entire real axis. For slope distribution x is a two-component vector consisting of the derivatives along the axes, and X is correspondingly the plane.

There are many more or less sophisticated entropy estimation methods. Gwyddion uses a relatively simple histogram based method in which the above formula is approximated with

$$S \approx -\sum_{i=1}^{n} p_i \log \frac{p_i}{w_i}$$

where p_i and w_i are the estimated probability density, i.e. normalized histogram value, and width of the i-th histogram bin.

Of course, the estimated entropy value depends on the choice of bin width. With the exception of uniform distribution, for which the estimate does not depend on the bin size. From this follows that for reasonable distributions a suitable bin width is such that the estimated entropy does not change when the width changes somewhat. This is the criterion used for choosing a suitable bin width.

In practice this means the entropy is estimated over a large range of bin widths (typically many orders of magnitude), obtaining the scaling curve displayed in the graph on the right side of the dialogue. The entropy is then estimated by finding the inflexion point of the curve. If there is no such point, which can happen for instance when the distribution is too close to a sum of δ-functions, a very large negative value is reported as the entropy.

The entropies are displayed as unitless since they are logarithmic quantities. For absolute comparison with other calculations, it should be noted that in Gwyddion they are always calculated from quantities in base SI units (e.g. metres as opposed to for instance nanometres) and natural logarithms are used. The values are thus in natural units of information (nats).

The absolute entropy depends on the absolute width of the distribution. For instance, two Gaussian distributions of heights with different RMS values have different entropies. This is useful when we want to compare the absolute narrowness of structures in the height distribution. On the other hand, it can be useful to compare them in a manner independent on the overall scale. For this we can utilise the fact that the Gaussian distribution has the maximum entropy among all distributions with the same RMS value. The difference between this maximum and the estimated entropy is displayed as entropy deficit. By definition, it is always positive (apart from numerical issues).

References

[1] D. Nečas and P. Klapetek: One-dimensional autocorrelation and power spectrum density functions of irregular regions. Ultramicroscopy 124 (2013) 13-19, doi:10.1016/j.ultramic.2012.08.002

[2] D. Nečas, P. Klapetek and M. Valtr: Estimation of roughness measurement bias originating from background subtraction. Measurement Science and Technology (2020) 094010, doi:10.1088/1361-6501/ab8993

[3] D. Nečas, M. Valtr and P. Klapetek: How levelling and scan line corrections ruin roughness measurement and how to prevent it. Scientific Reports 10 (2020) 15294, doi:10.1038/s41598-020-72171-8

[4] D. Nečas: Self-consistent autocorrelation for finite-area bias correction in roughness measurement. Engineering Research Express 6 (2024) 025560, doi:10.1088/2631-8695/ad5302

4.11 One-Dimensional Roughness Parameters

Standardized one-dimensional roughness parameters can be evaluated with the Roughness tool.

The one-dimensional texture is split into waviness (the low-frequency components defining the overall shape) and roughness (the high-frequency components) at the cut-off frequency. This frequency is specified in the units of the Nyquist frequency, that is value 1.0 corresponds to the Nyquist frequency. It is also displayed as the corresponding real-space wavelength.

In the following formulas we assume the mean value of r_j is zero, i.e. it holds

$$r_j = z_j - \bar{z}$$

Roughness Amplitude Parameters

Roughness Average R_a Standards: ASME B46.1-1995, ASME B46.1-1985, ISO 4287-1997, ISO 4287/1-1997.

Arithmetical mean deviation. The average deviation of all points roughness profile from a mean line over the evaluation length

$$R_a = \frac{1}{N}\sum_{j=1}^{N} |r_j|$$

An older means of specifying a range for R_a is RHR. This is a symbol on a drawing specifying a minimum and maximum value for R_a.

Root Mean Square Roughness R_q Standards: ASME B46.1-1995, ISO 4287-1997, ISO 4287/1-1997.

The average of the measured height deviations taken within the evaluation length and measured from the mean line

$$R_q = \sqrt{\frac{1}{N}\sum_{j=1}^{N} r_j^2}$$

Maximum Height of the Profile R_t Standards: ASME B46.1-1995, ISO 4287-1997.

Maximum peak-to-peak-valley height. The absolute value between the highest and lowest peaks

$$R_t = \left|\min_{1\le j\le N} r_j\right| + \left|\max_{1\le j\le N} r_j\right|$$

Maximum Profile Valley Depth R_v**,** R_m Standards: ASME B46.1-1995, ASME B46.1-1985, ISO 4287-1997, ISO 4287/1-1997.

Lowest valley. This is the depth of the deepest valley in the roughness profile over the evaluation length

$$R_v = \left|\min_{1\le j\le N} r_j\right|$$

Maximum Profile Peak Height R_p Standards: ASME B46.1-1995, ASME B46.1-1985, ISO 4287-1997, ISO 4287/1-1997.

Highest peak. This is the height of the highest peak in the roughness profile over the evaluation length

$$R_p = \left|\max_{1\le j\le N} r_j\right|$$

Average Maximum Height of the Profile R_{tm} Standards: ASME B46.1-1995, ISO 4287-1997.

Mean peak-to-valley roughness. It is determined by the difference between the highest peak ant the lowest valley within multiple samples in the evaluation length

$$R_{tm} = R_{vm} + R_{pm}$$

where R_{vm} and R_{pm} are defined below.

For profile data it is based on five sample lengths ($m = 5$). The number of samples corresponds with the ISO standard.

Average Maximum Profile Valley Depth $R_{\mathrm{v}m}$ Standards: ISO 4287-1997.

The mean valley depth based on one peak per sampling length. The single deepest valley is found in five sampling lengths ($m = 5$) and then averaged

$$R_{\mathrm{v}m} = \frac{1}{m}\sum_{i=1}^{m} R_{\mathrm{v}i}$$

where

$$R_{\mathrm{v}i} = \left|\min r_j\right| \quad \text{for} \quad (i-1)\frac{N}{m} < j < i\frac{N}{m}$$

Average Maximum Profile Peak Height $R_{\mathrm{p}m}$ Standards: ISO 4287-1997.

The mean peak height based on one peak per sampling length. The single highest peak is found in five sampling lengths ($m = 5$) and then averaged

$$R_{\mathrm{p}m} = \frac{1}{m}\sum_{i=1}^{m} R_{\mathrm{p}i}$$

where

$$R_{\mathrm{p}i} = \left|\max r_j\right| \quad \text{for} \quad (i-1)\frac{N}{m} < j < i\frac{N}{m}$$

Base roughness depth R_{3z} Standards: ISO 4287-1997.

The distance between the third highest peak and the third lowest valley. A peak is a portion of the surface above the mean line crossings.

Base roughness profile depth $R_{3z\mathrm{ISO}}$ Standards: ISO 4287-1997.

The height of the third highest peak from the third lowest valley per sampling length. The base roughness depth is found in five sampling lengths and then averaged.

Ten-point height R_z Standards: ISO 4287-1997

The average absolute value of the five highest peaks and the five lowest valleys over the evaluation length.

Average peak-to-valley profile roughness $R_{z\mathrm{ISO}}$ Standards: ISO 4287-1997.

The average peak-to-valley roughness based on one peak and one valley per sampling length. The single largest deviation is found in five sampling lengths and then averaged. It is identical to $R_{\mathrm{t}m}$.

The Amplitude Distribution Function ADF Standards: ISO 4287-1997.

The amplitude distribution function is a probability function that gives the probability that a profile of the surface has a certain height z at any position x.

The Bearing Ratio Curve BRC Standards: ISO 4287-1997.

The Bearing Ratio Curve is related to the ADF, it is the corresponding cumulative probability distribution and sees much greater use in surface finish. The bearing ratio curve is the integral (from the top down) of the ADF.

Skewness R_{sk} Standards: ISO 4287-1997.

Skewness is a parameter that describes the shape of the ADF. Skewness is a simple measure of the asymmetry of the ADF, or, equivalently, it measures the symmetry of the variation of a profile about its mean line

$$R_{\mathrm{sk}} = \frac{1}{NR_q^3}\sum_{j=1}^{N} r_j^3$$

Kurtosis R_{ku} Standards: ISO 4287-1997.

Kurtosis is the ADF shape parameter considered. Kurtosis relates to the uniformity of the ADF or, equivalently, to the spikiness of the profile.

$$R_{\mathrm{ku}} = \frac{1}{NR_q^4}\sum_{j=1}^{N} r_j^4$$

Swedish height H Height difference between two predefined lines. The upper line exposes 5 % of the surface and the lower line exposes 90 %. In other words it is the difference between 95th and 10th height distribution percentiles. H is less sensitive to data spikes than R_{t}.

4.12 Feature Measurement

Menu *Measure Features* offers several functions for the measurement of specific shapes or features in topographical images.

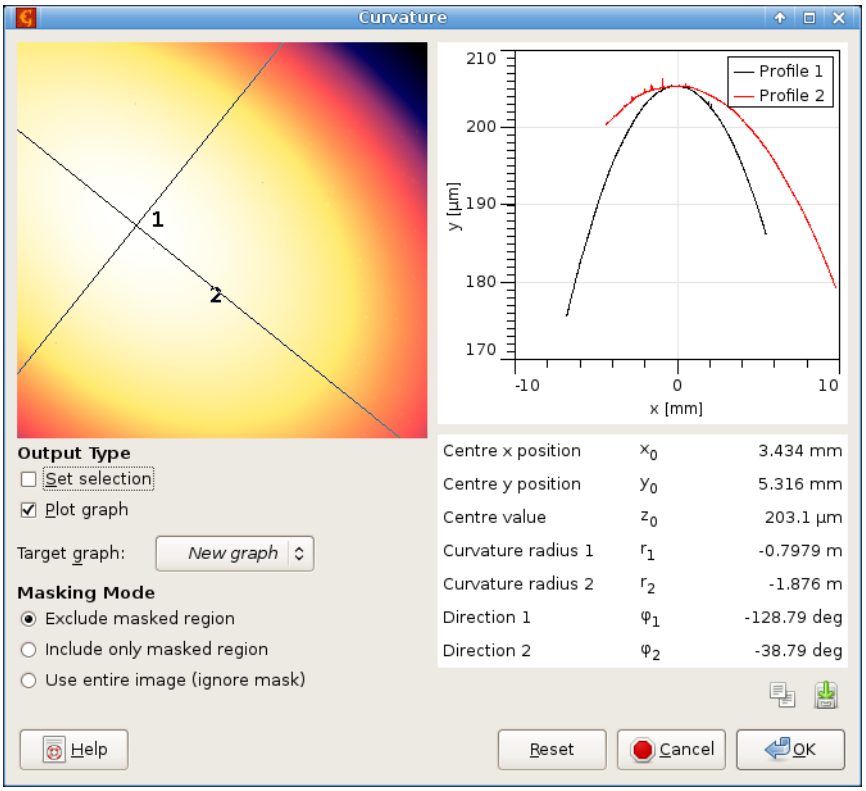
Curvature

Data Process → Measure Features → Curvature

The global surface curvature parameters are calculated by fitting a quadratic polynomial and finding its main axes. Positive signs of the curvature radii correspond to a concave (cup-like) surface, whereas negative signs to convex (cap-like) surface, mixed signs mean a saddle-like surface.

Beside the parameter table, it is possible to set the line selection on the data to the fitted quadratic surface axes and/or directly read profiles along them. The zero of the abscissa is placed to the intersection of the axes.

Similarly to the background subtraction functions, if a mask is present on the data the module offers to include or exclude the data under mask.



Curvature dialogue screenshot showing the strong deflection of a glass plate with a thin film with compressive internal stress.

Fit Shape

Data Process → Measure Features → Fit Shape

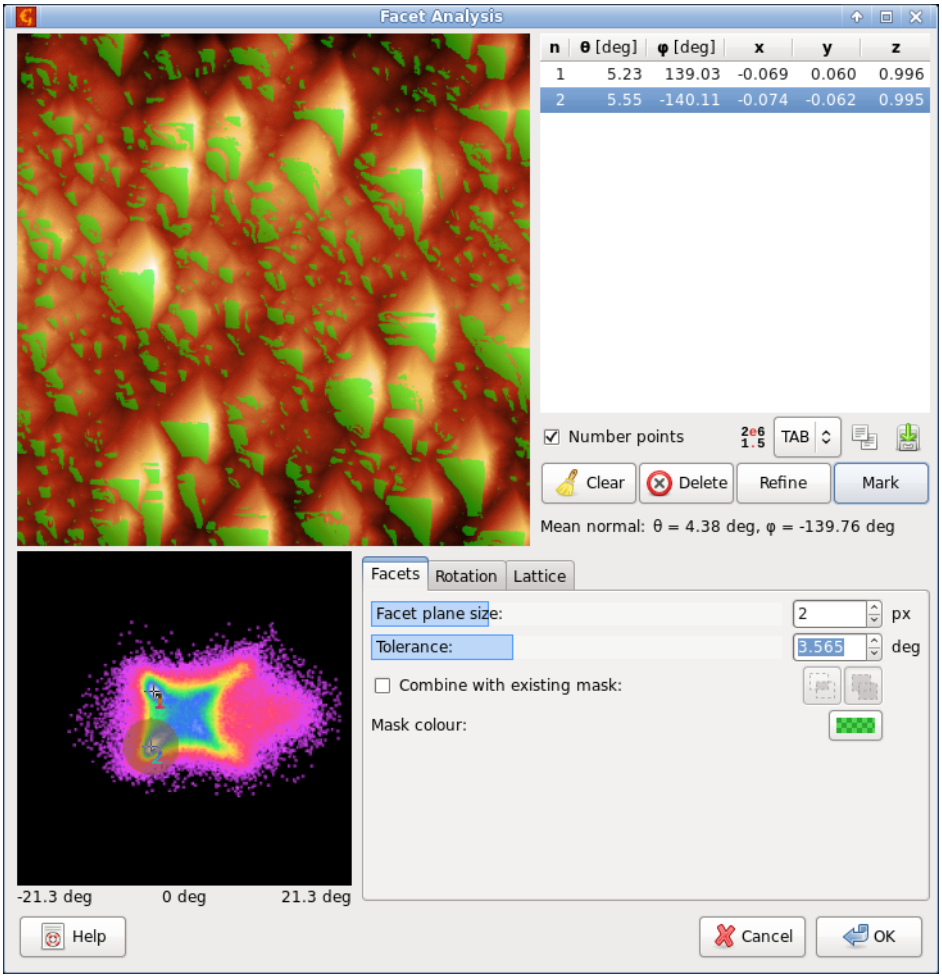
Form removal as well as measurement of geometrical parameters can be performed by fitting geometrical shapes on the data using the least-squares method. The *Fit Shape* module is essentially the same for images and XYZ data and it is described in detail in the XYZ data processing part. Only the differences are mentioned here.

Standard masking is supported for images. That is if a mask is present the dialogue offers to use the data under the mask, exclude the data under mask or ignore the mask and use the entire data. Since the excluded pixels can be outliers or image parts not conforming to the chosen shape it it possible to avoid them also in the difference image by disabling *Calculate differences for excluded pixels*.

Facet Analysis

Data Process → *Measure Features* → *Facet Analysis*

Facet analysis enables to interactively study orientations of facets occurring in the data and mark facets of specific orientations on the image. The top left view displays data with preview of marked facets. The smaller bottom left view, called facet view below, displays the two-dimensional slope distribution.



Facet analysis screenshot, showing two selected points on the angular distribution and marked facets for one of them.

The centre of facet view always correspond to zero inclination (horizontal facets), slope in x-direction increases towards left and right border and slope in y-direction increases towards top and bottom borders. The exact coordinate system is a bit complex (an area preserving transformation between spherical angles and the planar facet view is used) and it adapts to the range of slopes in the particular data displayed. The range of inclinations is displayed below the facet view.

Points selected on the facet view, i.e. directions in space, are listed in the top right corner as both angles and unit vector components. You can also choose the direction of the selected point by clicking on the image view. The direction is then set to the local facet at that position in the image.

Standard controls allowing manipulation with the entire list or the selected point are below the list. Button *Refine* tries to improve the position of selected point to a local maximum in the facet view. Button *Mark* creates a mask on the image corresponding to the selected direction and a range of close directions. The directions are simultaneously shown on the facet view. The mean normal of the selected region is displayed as *Mean normal* below.

The bottom right part contains tabs with settings and some advanced operations.

Facet *Facet plane size* controls the size (radius) of plane locally fitted in each point to determine the local inclination. The special value 0 stands for no plane fitting, the local inclination is determined from symmetric x and y derivatives in each

point. The choice of neighbourhood size is crucial for meaningful results: it must be smaller than the features one is interested in to avoid their smoothing, on the other hand it has to be large enough to suppress noise present in the image.

Tolerance determines the range of directions for marking. It also determines the search range for refining the the selected direction.

Rotation The set of selected directions can be rotated in space using either the shader control or by changing individual angles. The rotation angle around the vertical axis is denoted α and it is applied first. The directions are then rotated by angle ϑ in the direction given by azimuth φ.

The global rotation cannot be mixed with movement of individual points on the facet view. Once you start changing points there, the global rotation is reset and the current positions, whatever they are, become the initial positions. During rotation you can reset it to the starting state using the *Reset* button.

Lattice A set of directions corresponding to low-number Miller indices can be created by choosing the lattice type, its parameters and pressing button *Create points*. The absolute lengths of the cell edges are not important for the directions, only their ratios are. Therefore, they can be given in arbitrary units.

Together with rotation, this function can be useful for matching the image facets to crystallographic planes.

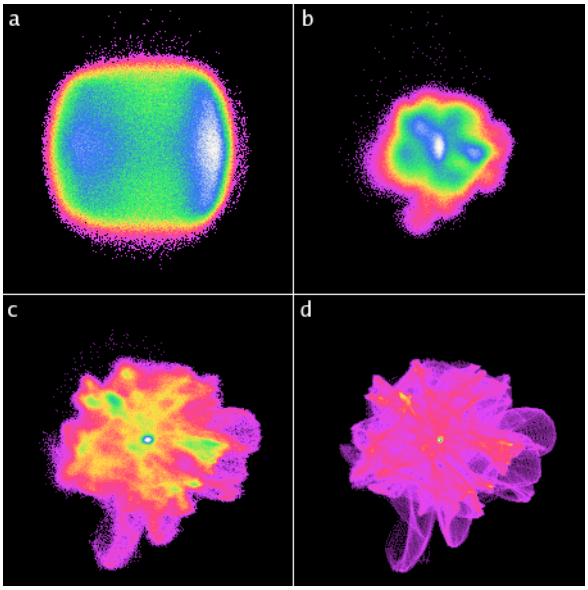


Illustration of the influence of fitted plane size on the distribution of a scan of a delaminated DLC surface with considerable fine noise. One can see the distribution is completely obscured by the noise at small plane sizes. The neighbourhood sizes are: (a) 0, (b) 2, (c) 4, (d) 7. The angle and false colour mappings are full-scale for each particular image, i.e. they vary among them.

Facet Measurement

Data Process → Measure Features → Facet Measurement

Facet measurement allows to interactively select and mark several facets of specific orientations on the image, measure their positions and store the results in a table. The top left view displays data with preview of marked facets. The smaller bottom left view, called facet view below, displays the two-dimensional slope distribution.

The centre of facet view always correspond to zero inclination (horizontal facets), slope in x-direction increases towards left and right border and slope in y-direction increases towards top and bottom borders. The exact coordinate system is a bit complex (an area preserving transformation between spherical angles and the planar facet view is used) and it adapts to the range of slopes in the particular data displayed. The range of inclinations is displayed below the facet view.

Facet plane size controls the size (radius) of plane locally fitted in each point to determine the local inclination. The special value 0 stands for no plane fitting, the local inclination is determined from symmetric x and y derivatives in each point. The choice of neighbourhood size is crucial for meaningful results: it must be smaller than the features one is interested in to avoid their smoothing, on the other hand it has to be large enough to suppress noise present in the image.

Tolerance determines the range of directions for marking. It also determines the search range for refining the the selected direction.

The currently selected point in facet view (facet orientation) is displayed as *Selected* ϑ and φ. Button *Refine* tries to improve the position of selected point to a local maximum in the facet view. Button *Mark* creates a mask on the image corresponding to the selected direction and a range of close directions. If *Instant facet marking* is enabled facets are marked immediately as the selected point changes (the *Mark* button is then disabled).

Once you are satisfied with the facet selection press *Measure* to add a measurement to the table below. Each row of the table contains

- Point index n.
- Number of pixels marked and used for the measurement.
- Tolerance t used in the measurement.
- Mean polar angle ϑ of the facet normal.
- Mean azimuthal angle φ of the facet normal.
- Facet normal as a unit vector (x, y, z)
- Standard deviation of facet normals within the marked set δ.

The mean normals are calculated by simple averaging of the unit normal vectors $\mathbf{n}$, which is appropriate when their spread is relatively small. The standard deviation is calculated in radians as follows

$$\delta^2 = \frac{1}{N-1}\sum_{i=1}^{N} |\mathbf{n}_i - \bar{\mathbf{n}}|^2$$

where bar denotes the mean normal.

Lattice 

Data Process → *Measure Features* → *Lattice*

Parameters of regular two-dimensional structures can be obtained by analysing data transformed into a form that captures their regularity. The ACF and PSDF are particularly well-suited for this because they exhibit peaks corresponding to the Bravais lattice vectors for the periodic pattern.

Peaks in the two-dimensional ACF correspond to lattice vectors in the direct space (and their integer multiples and linear combinations). Peaks in the two-dimensional PSDF correspond to lattice vectors of the *reciprocal* lattice. The matrices formed by the lattice vectors in direct and frequency space are transposed inverses of each other. With a suitable transformation either can be used to measure the lattice.

The Gwyddion function *Measure Lattice* can utilise either ACF or PSDF for the measurement. Since regular lattices with large periods correspond to peaks in the ACF image that are far apart but peaks in the PSDF that are close, it is preferable to use the ACF in this case to improve the resolution. Conversely, the PSDF image is more suitable if the lattice periods are small. In the dialogue you can switch freely between direct and frequency space representations and the selected lattice is transformed as necessary.

If scan lines contain lots of high-frequency noise the horizontal ACF (middle row in the ACF image) can be also quite noisy. In such case it is better to interpolate it from the surrounding rows by enabling the option *Interpolate horizontal ACF*.

The button *Estimate* attempts to find automatically the lattice for the image. When it does not select the vectors as you would like you can adjust the vectors in the image manually. The selection can be done either in a manner similar to Affine Distortion (this corresponds to choosing *Show lattice as Lattice*), or only the two vectors can be shown and adjusted (*Vectors*). Either way, when you choose approximately the peak positions, pressing the *Refine* button adjusts the positions of the vectors to improve the match with the peaks. Button *Reset* can be used to restore the vectors to a sane initial state.

The lattice vectors are always displayed as direct-space vectors, with components, total lengths and directions. The angle shown as φ is the angle between the two vectors.

The direct lattice vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ are related to the reciprocal lattice vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ by the following relation:

$$\mathbf{b}_1 = 2\pi\frac{R\mathbf{a}_2}{\mathbf{a}_1 \cdot R\mathbf{a}_2}\,, \qquad \mathbf{b}_2 = 2\pi\frac{R\mathbf{a}_1}{\mathbf{a}_2 \cdot R\mathbf{a}_1}\,,$$

where R denotes a 90 degree rotation matrix. The inverse transform is similar. Therefore, the lattice vectors must always be considered as a pair. Changing for instance $\mathbf{b}_1$ in the PSDF image changes both $\mathbf{a}_1$ and $\mathbf{a}_2$.

Sometimes it is useful to measure a single lattice vector, e.g. in images of one-dimensional periodic structures. The second vector then gets in the way. For this there exists a single-vector mode, enabled by *Measure single vector*. It hides the second vectors and ensures $\mathbf{a}_1$ are $\mathbf{b}_1$ are parallel and with mutually reciprocal lengths, as expected.

Terraces 

Data Process → Measure Features → Terraces

Terrace measurement can be used either to measure steps between plateaus in terrace or amphitheatre-like structures, or calibration by measuring such structures with precisely known step heights – in particular atomic steps on silicon [1].

The image processing has several parts:

1. edge detection using the Gaussian step filter,
2. segmentation into terraces,
3. fitting the image with polynomial background, allowing each terrace to be offset vertically differently,
4. estimation of step height and integer levels of individual terraces, and
5. fitting the image again, assuming given integer levels.

Some of them are optional. Furthermore, filtering and segmentation is interactive, whereas fitting is only run when button *Fit* is pressed. Intermediate results can be visualised by choosing the image to display using the *Display* menu. At the beginning, the most useful image is usually *Marked terraces* which shows individual recognised terraces in different colours.

Parameter *Step detection kernel* of edge detection is the same as for the Gaussian step filter. The following parameters control segmentation:

Step detection threshold Threshold for the edge-detection (as a percentage of maximum value of the filtered image). Too small threshold leads to fragmented terraces as too much of the image is considered edges. Too high threshold leads to merging of terraces which are actually at different heights – this is often best checked using the *Marked terraces* image.

Step broadening Regions recognised as edges are widened by given number of pixels. This helps with separation of terraces which should stay separate. In addition, it excludes regions too close to edges from polynomial fitting.

Minimum terrace area After segmentation, only sufficiently large terraces are kept. This option specifies the minimum terrace area, given as a fraction of the entire image area.

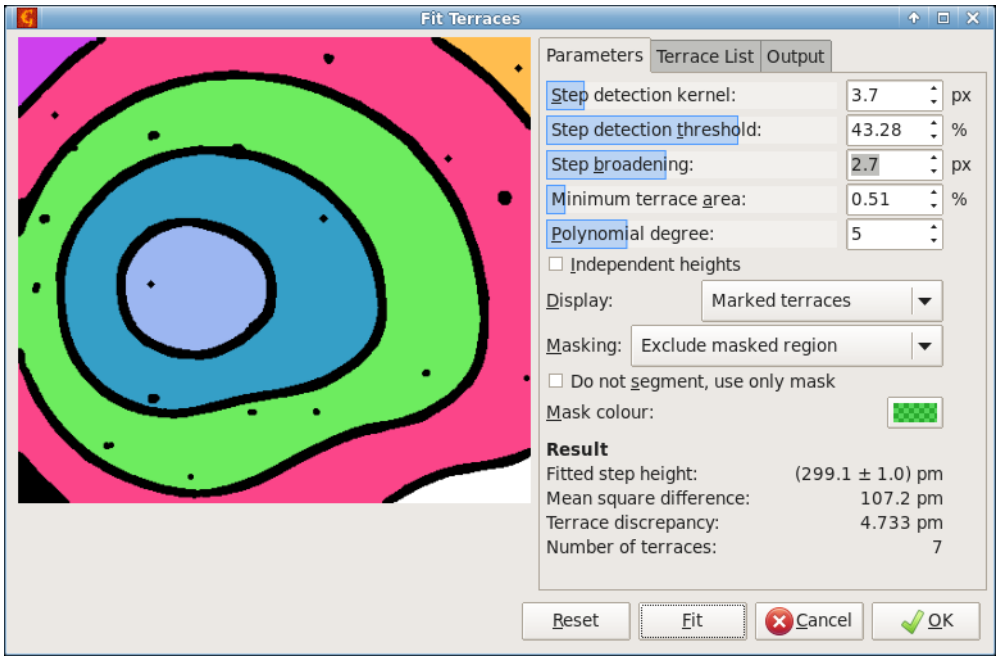
Segmentation can be augmented – or replaced – by a user-supplied mask. If the image has a mask, the standard masking options are offered. Here *Exclude masked region* means excluding it from the detected terraces. This allowing prevention of unwanted terrace merging – for instance by drawing manually small pieces of the boundaries using Mask Editor tool.

If option *Do not segment, use only mask* is enabled, no filtering is done and the mask is directly used as marked terraces (or their edges, depending on the masking mode). Filtering of terraces by area is still applied.

The total degree of the two-dimensional polynomial to fit is determined by parameter *Polynomial degree*. The polynomial is constructed in the same manner as in Polynomial levelling with limited total degree, except for the different base heights of each terrace.

The subsequent processing depends on whether *Independents heights* is enabled. If terrace heights are independent, each detected terrace is allowed to have arbitrary height offset. The module does not attempt to fit the data with one common step height. This is generally not useful with mono atomic steps which should be all of the same height, but can be useful for other types of terrace-like structures. The terrace list still shows estimated integer levels – which may or may not be meaningful in this case.

When heights are not independent, the module estimates the common step height and assigns an integer level to each terrace – this means the height of any terrace differs from any other by an integer multiple of the step height. Using this initial estimate, the polynomial fitting is performed again, leading to the final value of the step height.



Terrace measurement dialogue screenshot showing marked terraces for an amphitheatre structure formed by atomic steps on silicon.

The lower part of tab *Parameters* shows an overview of the fitting results:

- *Fitted step height*, usually the main result, shown with error obtained from the least-squares method,
- *Mean square difference* between the data and fit (including only terraces, not edges),
- *Terrace discrepancy*, the mean square difference between actual average terrace height and its height as a multiple of step height, and
- *Number of terraces* found, marked and used for the fitting.

Tab *Terraces* contains a table detailing properties of individual terraces. Each row corresponds to one terrace. The columns are:

- n, terrace index (and colour in *Marked terraces* image),
- h, average height – actual, not multiple of step height,
- k, integer level (multiple of step height),
- A_{px}, area in pixels,
- Δ, discrepancy, i.e. difference between average height and the assigned multiple of step height, and
- r, residuum, mean square difference between fit and data.

For relatively noisy data, discrepancies can be clearer indicators of correct fit than fit residua.

The resulting step height varies slightly with algorithm parameters. This dependence can be investigated using *Survey* by performing an automated scan over a range of polynomial degrees and/or step broadenings. All parameters not scanned in the survey are kept at values selected in the *Parameters* tab.

The survey is executed by pressing *Execute* and can take a while to finish. When it finishes a file dialogue will appear, allowing to choose the file where the resulting table should be saved.

References

[1] J. Garnæs, D. Nečas, L. Nielsen, M. Madsen, A. Torras-Rosell, G. Zeng, P. Klapetek, A. Yacoot, Algorithms for using silicon steps for scanning probe microscope evaluation. Metrologia 57 (2020) 064002, doi:10.1088/1681-7575/ab9ad3

4.13 Grain Analysis

There are several grain-related algorithms implemented in Gwyddion. First of all, simple thresholding algorithms can be used (height, slope or curvature thresholding). These procedures can be very efficient namely within particle analysis (to mark particles located on flat surface).

Thresholding 

Data Process → Grains → Mark by Threshold

Thresholding is the basic grain marking method. Height, slope and curvature thresholding is implemented within this module. The results of each individual thresholding methods can be merged together using logical operators.

Otsu's Method 

Data Process → Grains → Mark by Otsu's

The automated Otsu's thresholding method classifies the data values into two classes, minimising the intra-class variances within both. It is most suitable for images that contain two relatively well defined value levels.

Edge Detection 

Data Process → Grains → Mark by Edge Detection

Another grain marking function is based on edge detection (local curvature). The image is processed with a difference-of-Gaussians filter of a given size and thresholding is then performed on this filtered image instead of the original.

Remove Edge-Touching 

Data Process → Grains → Remove Edge-Touching

Grains that touch image edges can be removed using this function. It is useful if such grains are considered incomplete and must be excluded from analysis. Several other other functions that may be useful for modification of grain shapes after marking are performed by the Mask Editor tool.

Watershed 

Data Process → Grains → Mark by Watershed

For more complicated data structures the effectiveness of thresholding algorithms can be very poor. For these data a *watershed algorithm* can be used more effectively for grain or particle marking.

The watershed algorithm is usually employed for local minima determination and image segmentation in image processing. As the problem of determining the grain positions can be understood as the problem of finding local extremes on the surface this algorithm can be used also for purposes of grain segmentation or marking. For convenience in the following we will treat the data inverted in the z direction while describing the algorithm (i.e. the grain tops are forming local minima in the following text). We applied two stages of the grain analysis (see [1]):

1. Grain location phase: At each point of the inverted surface the virtual water drop was placed (amount of water is controlled by parameter *Drop size*). In the case that the drop was not already in a local minimum it followed the steepest descent path to minimize its potential energy. As soon as the drop reached any local minimum it stopped here and rested on the surface. In this way it filled the local minimum partially by its volume (see figure below and its caption). This process was repeated several times (parameter *Number of steps*). As the result a system of lakes of different sizes filling the inverted surface depressions was obtained. Then the area of each of the lakes was evaluated and the smallest lakes are removed under assumption that they were formed in the local minima originated by noise (all lakes smaller than parameter *Threshold* are removed). The larger lakes were used to identify the positions of the grains for segmentation in the next step. In this way the noise in the AFM data was eliminated. As a result
2. Segmentation phase: The grains found in the step 1 were marked (each one by a different number). The water drops continued in falling to the surface and filling the local minima (amount of water is controlled by parameter *Drop size*). Total number of steps of splashing a drop at every surface position is controlled by parameter *Number of steps*. As the grains were already identified and marked after the first step, the next five situations could happen as soon as the drop reached a local minimum.
   (a) The drop reached the place previously marked as a concrete grain. In this case the drop was merged with the grain, i. e. it was marked as a part of the same grain.
   (b) The drop reached the place where no grain was found but a concrete grain was found in the closest neighbourhood of the drop. In this case the drop was merged with the grain again.

(c) The drop reached the place where no grain was found and no grain was found even in the closest neighbourhood of the drop. In that case the drop was not marked at all.
(d) The drop reached the place where no grain was found but more than one concrete grain was found in the closest neighbourhood (e. g. two different grains were found in the neighbourhood). In this case the drop was marked as the grain boundary.
(e) The drop reached the place marked as grain boundary. In this case the drop was marked as the grain boundary too.

In this way we can identify the grain positions and then determine the volume occupied by each grain separately. If features of interest are valleys rather than grains (hills), parameter *Invert height* can be used.

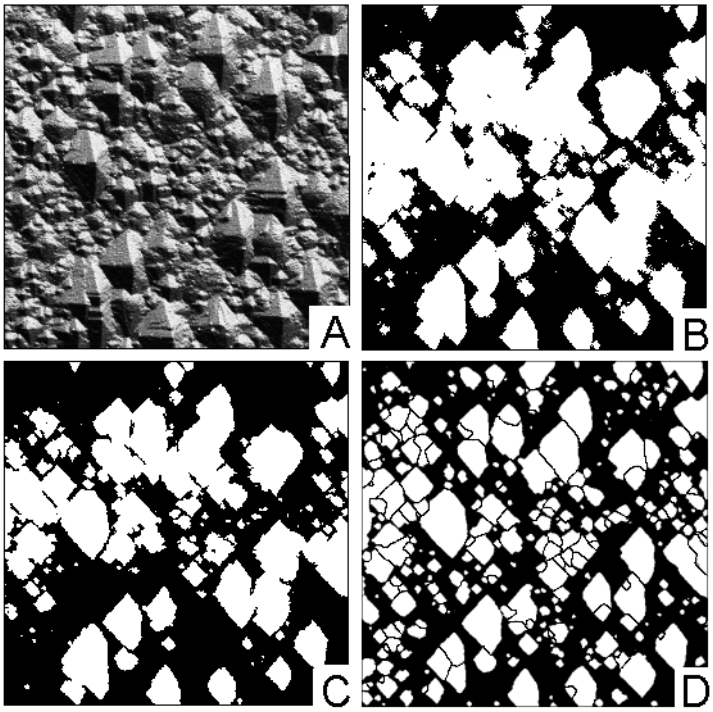


Image of grain-like surface structure (a) and corresponding results of height thresholding (b), curvature thresholding (c), and watershed (d) algorithm. Within watershed algorithm it is possible to segment image even further.

Segmentation

Data Process → *Grains* → *Mark by Segmentation*

This function a different approach based on a *watershed algorithm*, in this case the classical Vincent algorithm for watershed in digital spaces [2], which is applied to a preprocessed image. Generally, the result is an image fully segmented to motifs, each pixel belonging to one or separating two of them. By default, the algorithm marks *valleys*. To mark upward grains, which is more common in AFM, use the option *Invert height*.

The preprocessing has the following parameters:

Gaussian smoothing Dispersion of Gaussian smoothing filter applied to the data. A zero value means no smoothing.

Add gradient Relative weight of local gradient added to the data. Large values mean areas with large local slope tend to become grain boundaries.

Add curvature Relative weight of local gradient added to the data. Large values mean locally concave areas with tend to become grain boundaries.

Barrier level Relative height level above which pixels are never assigned to any grain. If not 100%, this creates an exception to the full-segmentation property.

Prefill level Relative height level up to which the surface is prefilled, obliterating any details at the bottoms of deep valleys.

Prefill from minima Relative height level up to which the surface is prefilled from each local minimum, obliterating any details at the bottoms of valleys.

Logistic Regression

Data Process → Grains → Logistic Regression

To transfer segmentation from one image to another one with similar features, or from small chunk to complete large image, the logistic regression module can be used. It has two modes of operation, in training phase it requires the mask of grains to be present on the image, and train logistic regression for this mask. When the regression is trained, one can use it for the new images. In both modes it generates a feature vector from the current image, applying a number of filters to it. The available features include:

Gaussian blur Gaussian filters with the length of 2, 4, 8 and so on. The number of gaussians can be selected by the user with *Number of Gaussians*

Sobel derivatives Sobel first derivative filters for X and Y directions.

Laplacian Laplacian sum of second deriatives operator.

Hessian three elements of Hessian matrix of second derivatives, one for each of X- and Y-directions and one mixed.

All derivative filters if selected are applied to original image as well as to each of Gaussian-filtered if they present. Each feature layer is computed with the convolution operator and than normalized.

Regularization parameter allows to regularize logistic regression, larger values means that parameters are more restricted from having very large values.

Note, that you need to have a mask on image for training, or the training will give zero results. The training phase can be very slow, especially for large images and large set of features selected. The training results are saved between the invocation, so they can be applied to other images. If the set of features is modified, one need to train logistic regression from the scratch.

Statistics

Grain properties can be studied using several functions. The simplest of them is Grain Summary.

Grain Summary

Data Process → Grains → Summary

This function calculates the total number of marked grains, their total projected area, both as an absolute value and as a fraction of total data field area, total grain volumes, total length of grain boundaries and the mean area and equivalent square size of one grain. The mean size is calculated by averaging the equivalent square sides so its square is not, in general, equal to the mean area.

Overall characteristics of the marked area can be also obtained with Statistical Quantities tool when its *Use mask* option is switched on. By inverting the mask the same information can be obtained also for the non-grain area.

Grain Statistics

Data Process → Grains → Statistics

The grain statistics function displays the mean, median, standard deviation (rms) and inter-quartile range for all available grain quantities in one table.

Grain Distributions 

Data Process → Grains → Distributions

Grain Distributions is the most powerful and complex tool. It has two basic modes of operation: graph plotting and raw data export. In graph plotting mode selected characteristics of individual grains are calculated, gathered and summary graphs showing their distributions are plotted.

Raw data export is useful for experts who need for example to correlate properties of individual grains. In this mode selected grain characteristics are calculated and dumped to a text file table where each row corresponds to one grain and columns correspond to requested quantities. The order of the columns is the same as the relative order of the quantities in the dialogue; all values are written in base SI units, as is usual in Gwyddion.

Grain Property Correlation 

Data Process → Grains → Correlate

Grain correlation plots a graph of one selected graph quantity as the function of another grain quantity, visualizing correlations between them.

Grain Measurement Tool 

The grain measurement tool is the interactive method to obtain the same information about individual grains as Grain Distributions in raw mode. After selecting a grain on the data window with mouse, all the available quantities are displayed in the tool window.

Beside physical characteristics this tool also displays the grain number. Grain numbers corresponds to row numbers (counting from 1) in files exported by Grain Distributions.

Grain Properties

Grain Distributions and Grain measurement tool can calculate the following grain properties:

Value-related properties

- *Minimum*, the minimum value (height) occurring inside the grain.
- *Maximum*, the maximum value (height) occurring inside the grain.
- *Mean*, the mean of all values occurring inside the grain, that is the mean grain height.
- *Median* the median of all values occurring inside the grain, that is the median grain height.
- *RMS* the standard deviation of all values occurring inside the grain.
- *Minimum on boundary*, the minimum value (height) occurring on the inner grain boundary. This means within the set of pixels that lie inside the grain but at least one of their neighbours lies outside.
- *Maximum on boundary*, the maximum value (height) occurring on the inner grain boundary, defined similarly to the minimum.

Area-related properties

- *Projected area*, the projected (flat) area of the grain.
- *Equivalent square side*, the side of the square with the same projected area as the grain.
- *Equivalent disc radius*, the radius of the disc with the same projected area as the grain.
- *Surface area*, the surface area of the grain, see statistical quantities section for description of the surface area estimation method.
- *Area of convex hull*, the projected area of grain convex hull. The convex hull area is slightly larger than the grain area even for grains that appear to be fairly convex due to pixelisation of the mask. The only grains with exactly the same area as their convex hulls are perfectly rectangular grains.

Boundary-related properties

- *Projected boundary length*, the length of the grain boundary projected to the horizontal plane (that is not taken on the real three-dimensional surface). The method of boundary length estimation is described below.
- *Minimum bounding size*, the minimum dimension of the grain in the horizontal plane. It can be visualized as the minimum width of a gap in the horizontal plane the grain could pass through.
- *Minimum bounding direction*, the direction of the gap from the previous item. If the grain exhibits a symmetry that makes several directions to qualify, an arbitrary direction is chosen.
- *Maximum bounding size*, the maximum dimension of the grain in the horizontal plane. It can be visualized as the maximum width of a gap in the horizontal plane the grain could fill up.
- *Maximum bounding direction*, the direction of the gap from the previous item. If the grain exhibits a symmetry that makes several directions to qualify, an arbitrary direction is chosen.
- *Maximum inscribed disc radius*, the radius of maximum disc that fits inside the grain. The entire full disc must fit, not just its boundary circle, which matters for grains with voids within. You can use Mask Editor tool to fill voids in grains to get rid of voids.
- *Maximum inscribed disc centre x position*, the horizontal coordinate of centre of the maximum inscribed disc. More precisely, of one such disc if it is not unique.
- *Maximum inscribed disc centre y position*, the vertical coordinate of centre of the maximum inscribed disc. More precisely, of one such disc if it is not unique but of the same as in the previous item.
- *Minimum circumcircle radius*, the radius of minimum circle that contains the grain.
- *Minimum circumcircle centre x position*, the horizontal coordinate of centre of the minimum circumcircle.
- *Minimum circumcircle centre y position*, the vertical coordinate of centre of the minimum circumcircle.
- *Mean radius*, the mean distance from grain centre of mass to its boundary. This quantity is mostly meaningful only for convex or nearly-convex grains.
- *Minimum Martin diameter*, the minimum length of a median line through the grain.
- *Direction of minimum Martin diameter*, the direction of the line from the previous item. If the grain exhibits a symmetry that makes several directions to qualify, an arbitrary direction is chosen.

- *Maximum Martin diameter*, the maximum length of a median line through the grain.
- *Direction of maximum Martin diameter*, the direction of the line from the previous item. If the grain exhibits a symmetry that makes several directions to qualify, an arbitrary direction is chosen.

Volume-related properties

- *Zero basis*, the volume between grain surface and the plane $z = 0$. Values below zero form negative volumes. The zero level must be set to a reasonable value (often Fix Zero is sufficient) for the results to make sense, which is also the advantage of this method: one can use basis plane of his choice.
- *Grain minimum basis*, the volume between grain surface and the plane $z = z_{\min}$, where $z_{\min}$ is the minimum value (height) occuring in the grain. This method accounts for grain surrounding but it typically underestimates the volume, especially for small grains.
- *Laplacian backround basis*, the volume between grain surface and the basis surface formed by laplacian interpolation of surrounding values. In other words, this is the volume that would disappear after using Interpolate Data Under Mask or Grain Remover tool with Laplacian interpolation on the grain. This is the most sophisticated method, on the other hand it is the hardest to develop intuition for.

Position-related properties

- *Centre x position*, the horizontal coordinate of the grain centre. Since the grain area is defined as the area covered by the corresponding mask pixels, the centre of a single-pixel grain has half-integer coordinates, not integer ones. Data field origin offset (if any) is taken into account.
- *Centre y position*, the vertical coordinate of the grain centre. See above for the interpretation.

Slope-related properties

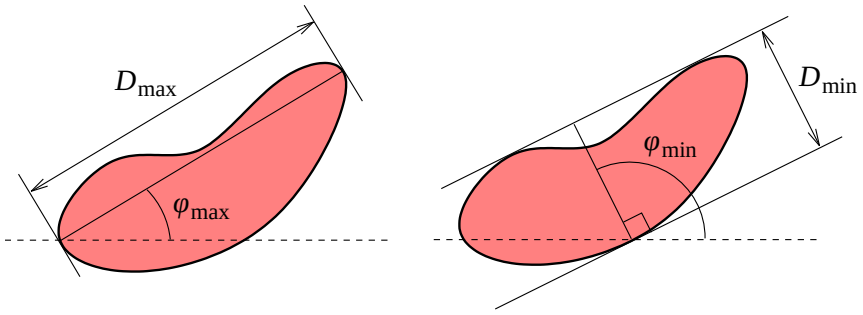
- *Inclination* ϑ, the deviation of the normal to the mean plane from the z-axis, see inclinations for details.
- *Inclination* φ, the azimuth of the slope, as defined in inclinations.

Curvature-related properties

- *Curvature centre x*, the horizontal position of the centre of the quadratic surface fitted to the grain surface.
- *Curvature centre y*, the vertical position of the centre of the quadratic surface fitted to the grain surface.
- *Curvature centre z*, the value at the centre of the quadratic surface fitted to the grain surface. Note this is the value at the fitted surface, not at the real grain surface.
- *Curvature 1*, the smaller curvature (i.e. the inverse of the curvature radius) at the centre.
- *Curvature 2*, the larger curvature (i.e. the inverse of the curvature radius) at the centre.
- *Curvature angle 1*, the direction corresponding to *Curvature 1*.
- *Curvature angle 2*, the direction corresponding to *Curvature 2*.

Moment-related properties

- *Major semiaxis of equivalent ellipse*, the length of the major semiaxis of an ellipse which has the same second order moments in the horizontal plane.
- *Minor semiaxis of equivalent ellipse*, the length of the minor semiaxis of an ellipse which has the same second order moments in the horizontal plane.
- *Orientation of equivalent ellipse*, the direction of the major semiaxis of an ellipse which has the same second order moments in the horizontal plane. For a circular grain, the angle is set to zero by definition.



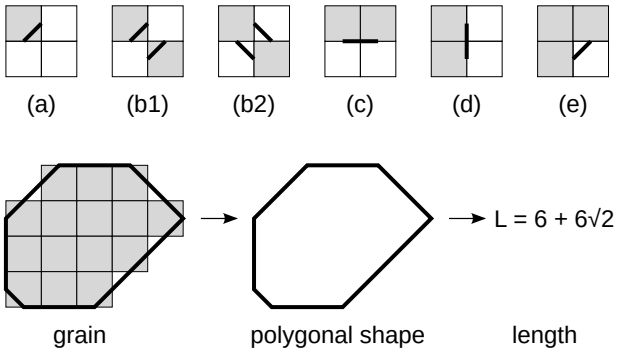
Maximum and minimum bounding dimensions and angles of a grain.

The grain boundary length is estimated by summing estimated contributions of each four-pixel configuration on the boundary. The contributions are displayed on the following figure for each type of configuration, where h_x and h_y are pixel dimension along

corresponding axes and h is the length of the pixel diagonal:

$$h = \sqrt{h_x^2 + h_y^2}$$

The contributions correspond one-to-one to lengths of segments of the boundary of a polygon approximating the grain shape. The construction of the equivalent polygonal shape can also be seen in the figure.



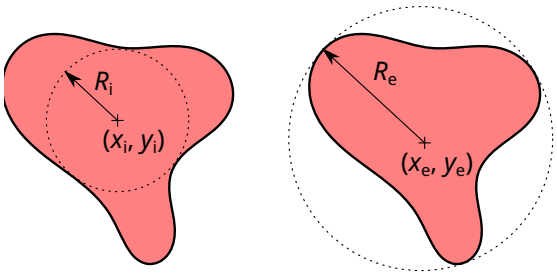
Contributions of pixel configurations to the estimated boundary length (top). Grey squares represent pixels inside the grain, white squares represent outside pixels. The estimated contribution of each configuration is: (a) $h/2$, (b1), (b2) h, (c) h_y, (d) h_x, (e) $h/2$. Cases (b1) and (b2) differ only in the visualization of the polygonal shape segments, the estimated boundary lengths are identical. The bottom part of the figure illustrates how the segments join to form the polygon.

The grain volume is, after subtracting the basis, estimated as the volume of exactly the same body whose upper surface is used for surface area calculation. Note for the volume between vertices this is equivalent to the classic two-dimensional trapezoid integration method. However, we calculate the volume under a mask centred on vertices, therefore their contribution to the integral is distributed differently as shown in the following figure.

$\frac{1}{96}$	$\frac{5}{48}$	$\frac{1}{96}$
$\frac{5}{48}$	$\frac{13}{24}$	$\frac{5}{48}$
$\frac{1}{96}$	$\frac{5}{48}$	$\frac{1}{96}$

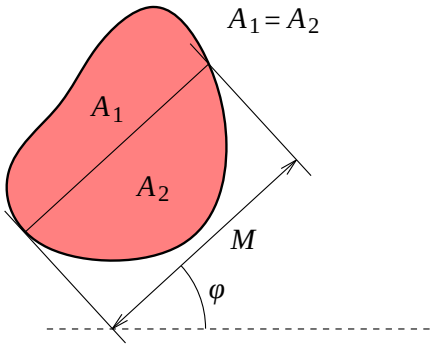
Contributions of individual pixels to the volume of single pixel (grey).

Curvature-related properties of individual grains are calculated identically to the global curvature calculated by Curvature. See its description for some discussion.



Maximum inscribed disc and minimum circumcircle of a grain.

Inscribed discs and circumcircles of grains can be visualized using *Data Process* → *Grains* → *Select Inscribed Discs* and *Data Process* → *Grains* → *Select Circumscribed Circles*. These functions create circular selections representing the corresponding disc or circle for each grain that can be subsequently displayed using Selection Manager tool.



Determination of Martin's diameter at given angle.

The Martin's diameter is the length of the unique line through the grain in given direction which which divides the grain area into two equal parts (median line). This quantity is mostly useful for grains that are still relatively close to convex. For reference we should note that if the grain is highly non-convex and the median line intersects the grain several times, Gwyddion sums the lengths of the segments actually lying inside the grain (the other option would be to simply take the distance between the first and last intersection).

Grain Filtering 

Marked grains can be filtered by thresholding by any of the available grain quantities using *Data Process* → *Grains* → *Filter* menu choice. The module can be used for basic operations, such as removal of tiny grains using a pixel area threshold, as well as complex filtering using logical expressions involving several grain quantities.

The filter retains grains that satisfy the condition specified as *Keep grains satisfying* and removes all other grains. The condition is expressed as a logical expression of one to three individual thresholding conditions, denoted `A`, `B` and `C`. The simplest expression is just `A`, stating that quantity `A` must lie within given thresholds.

Each condition consits of lower and upper thresholds for one grain quantity, for instance pixel area or minimum value. The values must lie within the interval [lower,upper] to satisfy the condition and thus retain the grains. Note it is possible to choose the lower threshold larger than the upper threshold. In this case the condition is inverted, i.e. the grain is retained if the value lies outside [upper,lower].

Individual grain quantities are assigned to `A`, `B` and `C` by selecting the quantity in the list and clicking on the corresponding button in *Set selected as*. The currently selected set of quantities is displayed in the *Condition A*, *Condition B* and *Condition C* headers.

Grain Number Extraction

Data Process → *Grains* → *Exract Numbers*

When images with marked grains are further processed in other software, it can be useful to clearly identify which region corresponds has which grain number assigned in Gwyddion. This function creates an image where each pixel contains the corresponding grain number. Hence, even though the data are still floating point numbers all the values are in fact integers.

Grain Leveling

Grains can be aligned vertically using *Data Process* → *Grains* → *Level Grains*. This function vertically shifts each grain to make a certain height-related quantity of all grains equal. Typically, the grain minimum values are aligned but other choices are possible.

Data between grains are also vertically shifted. The shifts are interpolated from the grain shifts using the Laplace equation, leading to a smooth transition of the shifts between the grains (though with no regard to other possible surface features).

References

[1] P. Klapetek, I. Ohlídal, D. Franta, A. Montaigne-Ramil, A. Bonanni, D. Stifter, H. Sitter: Atomic force microscopy characterization of ZnTe epitaxial films. Acta Physica Slovaca 53 (2003) 223–230, doi:10.1143/JJAP.42.4706

[2] Luc Vincent and Pierre Soille: Watersheds in digital spaces: an efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13 (1991) 583–598, doi:10.1109/34.87344

4.14 Fourier Transform

Two-dimensional Fourier transform can be accessed using *Data Process* → *Integral Transforms* → *2D FFT* which implements the Fast Fourier Transform (FFT). Fourier transform decomposes signal into its harmonic components, it is therefore useful while studying spectral frequencies present in the SPM data.

The *2D FFT* module provides several types of output:

- Modulus – absolute value of the complex Fourier coefficient, proportional to the square root of the power spectrum density function (PSDF).
- Phase – phase of the complex coefficient (rarely used).
- Real – real part of the complex coefficient.
- Imaginary – imaginary part of the complex coefficient.

and some of their combinations for convenience.

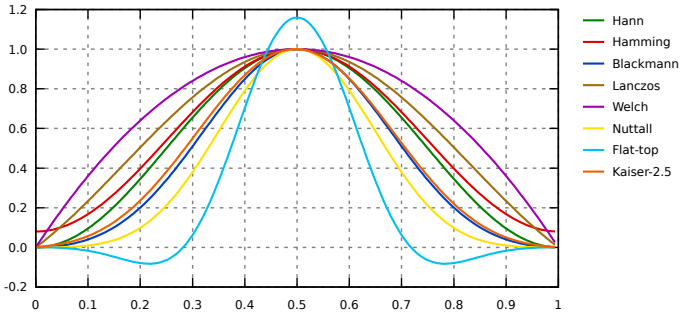
Radial sections of the two-dimensional PSDF can be conveniently obtained with 2D PSDF. Several other functions producing spectral densities are described in section Statistical Analysis. It is also possible to filter images in the frequency domain using one-dimensional and two-dimensional FFT filters or simple frequency splitting.

For scale and rotation-independent texture comparison it is useful to transform the PSDF from Cartesian frequency coordinates to coordinates consisting of logarithm of the spatial frequency and its direction. Scaling and rotation then become mere shifts in the new coordinates. The function *Data Process* → *Statistics* → *Log-Phi PSDF* calculates directly the transformed PSDF. The dimensionless horizontal coordinate is the angle (from 0 to 2π) the vertical is the frequency logarithm. It is possible to smooth the PSDF using a Gaussian filter of given width before the transformation.

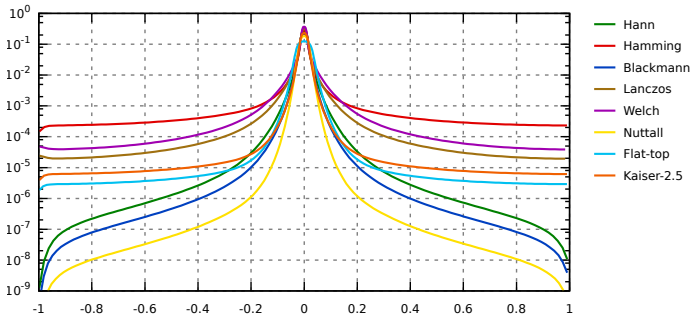
Note that the Fourier transform treats data as being infinite, thus implying some cyclic boundary conditions. As the real data do not have these properties, it is necessary to use some windowing function to suppress the data at the edgest of the image. If you do not do this, FFT treats data as being windowed by rectangular windowing function which has really bad Fourier image thus leading to corruption of the Fourier spectrum.

Gwyddion offers several windowing functions. Most of them are formed by some sine and cosine functions that damp data correctly at the edges. In the following windowing formula table the independent variable x is from interval $[0,1]$ which corresponds to the normalized abscissa; for simplicity variable $\xi = 2\pi x$ is used in some formulas. The available windowing types include:

Name	Formula
None	1
Rect	0.5 at edge points, 1 everywhere else
Hann	$w_{\text{Hann}}(x) = 0.5 - 0.5\cos\xi$
Hamming	$w_{\text{Hamming}}(x) = 0.54 - 0.46\cos\xi$
Blackmann	$w_{\text{Blackmann}}(x) = 0.42 - 0.5\cos\xi + 0.08\cos 2\xi$
Lanczos	$w_{\text{Lanczos}}(x) = \operatorname{sinc}\pi(2x-1)$
Welch	$w_{\text{Welch}}(x) = 4x(1-x)$
Nutall	$w_{\text{Nutall}}(x) = 0.355768 - 0.487396\cos\xi + 0.144232\cos 2\xi - 0.012604\cos 3\xi$
Flat-top	$w_{\text{flattop}}(x) = 0.25 - 0.4825\cos\xi + 0.3225\cos 2\xi - 0.097\cos 3\xi + 0.008\cos 4\xi$
Kaiser α	$w_{\text{Kaiser},\alpha}(x) = \dfrac{\mathrm{I}_0\left(\pi\alpha\sqrt{4x(1-x)}\right)}{\mathrm{I}_0(\pi\alpha)}$, where I_0 is the modified Bessel function of zeroth order and α is a parameter



Windowing functions: Hann, Hamming, Blackmann, Lanczos, Welch, Nutall, Flat-top, Kaiser 2.5.



Envelopes of windowing functions frequency responses: Hann, Hamming, Blackmann, Lanczos, Welch, Nutall, Flat-top, Kaiser 2.5.

1D FFT Filter

Data Process → Correct Data → 1D FFT Filtering

One excellent way of removing frequency based of noise from an image is to use Fourier filtering. First, the Fourier transform of the image is calculated. Next, a filter is applied to this transform. Finally, the inverse transform is applied to obtain a filtered image. Gwyddion uses the Fast Fourier Transform (or FFT) to make this intensive calculation much faster.

Within the 1D FFT filter the frequencies that should be removed from spectrum (suppress type: null) or suppressed to value of neighbouring frequencies (suppress type: suppress) can be selected by marking appropriate areas in the power spectrum graph. The selection can be inverted easily using the Filter type choice. 1D FFT filter can be used both for horizontal and vertical direction.

2D FFT Filter

Data Process → Correct Data → 2D FFT Filtering

2D FFT filter acts similarly as the 1D variant (see above) but using 2D FFT transform. Therefore, the spatial frequencies that should be filtered must be selected in 2D using mask editor. As the frequencies are related to center of the image (corresponding to zero frequency), the mask can be snapped to the center (coordinate system origin) while being edited. There are also different display and output modes that are self-explanatory – image or FFT coefficients can be outputted by module (or both).

Frequency Split

Data Process → Level → Frequency Split

A simple alternative to 2D FFT filtering is the separation of low and high spatial frequencies in the image using a simple low-pass and/or high-pass filter. The frequency split module can create either the low-frequency or high-frequency image or both, depending on the selected *Output type*.

Cut-off selects the spatial frequency cut off, which is displayed as relative fraction of the Nyquist frequency and also as the corresponding spatial wavelength. If *Edge width* (again given as relative to Nyquist frequency) is zero the filter has sharp edge. For non-zero values the transition has the shape of error function, with specified width.

Frequency domain filtering can lead to artefacts close to image boundaries. Therefore, several boundary handling options are provided, aside from *None* which is mostly useful for periodic data (or otherwise continuous across edges). *Laplace* and *Mirror* extend the image by Laplace's equation method and mirroring, respectively, exactly as the Extend function. *Smooth connect* applies to rows and column the one-dimensional smooth extension method used in Roughness tool to suppress edge artefacts.

4.15 Wavelet Transform

The wavelet transform is similar to the Fourier transform (or much more to the windowed Fourier transform) with a completely different merit function. The main difference is this: Fourier transform decomposes the signal into sines and cosines, i.e. the functions localized in Fourier space; in contrary the wavelet transform uses functions that are localized in both the real and Fourier space. Generally, the wavelet transform can be expressed by the following equation:

$$F(a,b) = \int_{-\infty}^{\infty} f(x)\, \psi^*_{(a,b)}(x)\, \mathrm{d}x$$

where the * is the complex conjugate symbol and function ψ is some function. This function can be chosen arbitrarily provided that it obeys certain rules.

As it is seen, the Wavelet transform is in fact an infinite set of various transforms, depending on the merit function used for its computation. This is the main reason, why we can hear the term "wavelet transform" in very different situations and applications. There are also many ways how to sort the types of the wavelet transforms. Here we show only the division based on the wavelet orthogonality. We can use *orthogonal wavelets* for discrete wavelet transform development and *non-orthogonal wavelets* for continuous wavelet transform development. These two transforms have the following properties:

1. The discrete wavelet transform returns a data vector of the same length as the input is. Usually, even in this vector many data are almost zero. This corresponds to the fact that it decomposes into a set of wavelets (functions) that are orthogonal to its translations and scaling. Therefore we decompose such a signal to a same or lower number of the wavelet coefficient spectrum as is the number of signal data points. Such a wavelet spectrum is very good for signal processing and compression, for example, as we get no redundant information here.
2. The continuous wavelet transform in contrary returns an array one dimension larger than the input data. For a 1D data we obtain an image of the time-frequency plane. We can easily see the signal frequencies evolution during the duration of the signal and compare the spectrum with other signals spectra. As here is used the non-orthogonal set of wavelets, data are highly correlated, so big redundancy is seen here. This helps to see the results in a more humane form.

For more details on wavelet transform see any of the thousands of wavelet resources on the Web, or for example [1].

Within Gwyddion data processing library, both these transforms are implemented and the modules using wavelet transforms can be accessed within *Data Process* → *Integral Transforms* menu.

Discrete Wavelet Transform 

The discrete wavelet transform (DWT) is an implementation of the wavelet transform using a discrete set of the wavelet scales and translations obeying some defined rules. In other words, this transform decomposes the signal into mutually orthogonal set of wavelets, which is the main difference from the continuous wavelet transform (CWT), or its implementation for the discrete time series sometimes called discrete-time continuous wavelet transform (DT-CWT).

The wavelet can be constructed from a scaling function which describes its scaling properties. The restriction that the scaling functions must be orthogonal to its discrete translations implies some mathematical conditions on them which are mentioned everywhere, e.g. the dilation equation

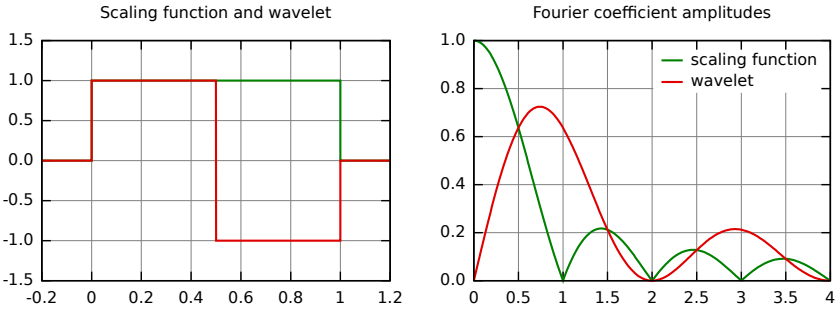
$$\phi(x) = \sum_{k=-\infty}^{\infty} a_k \phi(Sx - k)$$

where S is a scaling factor (usually chosen as 2). Moreover, the area between the function must be normalized and scaling function must be orthogonal to its integer translations, i.e.

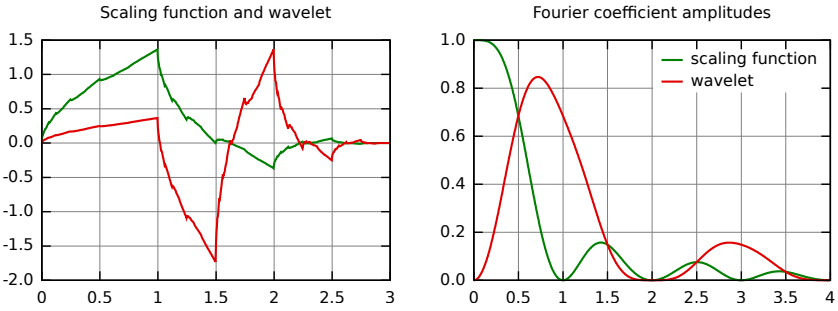
$$\int_{-\infty}^{\infty} \phi(x)\,\phi(x+l)\,\mathrm{d}x = \delta_{0,l}$$

After introducing some more conditions (as the restrictions above does not produce a unique solution) we can obtain results of all these equations, i.e. the finite set of coefficients a_k that define the scaling function and also the wavelet. The wavelet is obtained from the scaling function as N where N is an even integer. The set of wavelets then forms an orthonormal basis which we use to decompose the signal. Note that usually only few of the coefficients a_k are nonzero, which simplifies the calculations.

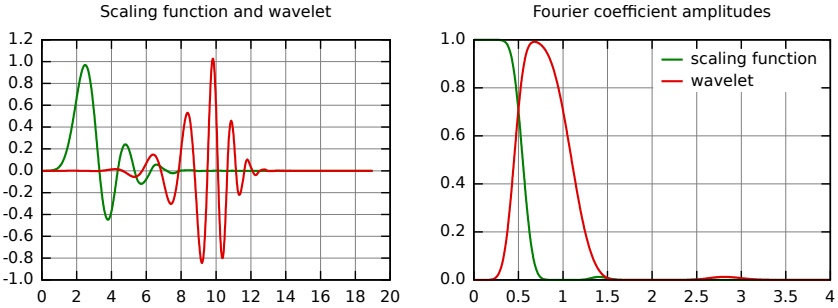
In the following figure, some wavelet scaling functions and wavelets are plotted. The most known family of orthonormal wavelets is the family of Daubechies. Her wavelets are usually denominated by the number of nonzero coefficients a_k, so we usually talk about Daubechies 4, Daubechies 6, etc. wavelets. Roughly said, with the increasing number of wavelet coefficients the functions become smoother. See the comparison of wavelets Daubechies 4 and 20 below. Another mentioned wavelet is the simplest one, the Haar wavelet, which uses a box function as the scaling function.



Haar scaling function and wavelet (left) and their frequency content (right).



Daubechies 4 scaling function and wavelet (left) and their frequency content (right).



Daubechies 20 scaling function and wavelet (left) and their frequency content (right).

There are several types of implementation of the DWT algorithm. The oldest and most known one is the Mallat (pyramidal) algorithm. In this algorithm two filters – smoothing and non-smoothing one – are constructed from the wavelet coefficients and those filters are recurrently used to obtain data for all the scales. If the total number of data $D = 2^N$ is used and the signal length is L, first $D/2$ data at scale $L/2^{N-1}$ are computed, then $(D/2)/2$ data at scale $L/2^{N-2}$, ... up to finally obtaining 2 data at scale $L/2$. The result of this algorithm is an array of the same length as the input one, where the data are usually sorted from the largest scales to the smallest ones.

Within Gwyddion the pyramidal algorithm is used for computing the discrete wavelet transform. Discrete wavelet transform in 2D can be accessed using DWT module.

Discrete wavelet transform can be used for easy and fast denoising of a noisy signal. If we take only a limited number of highest coefficients of the discrete wavelet transform spectrum, and we perform an inverse transform (with the same wavelet basis) we can obtain more or less denoised signal. There are several ways how to choose the coefficients that will be kept. Within Gwyddion, the universal thresholding, scale adaptive thresholding [2] and scale and space adaptive thresholding [3] is implemented. For threshold determination within these methods we first determine the noise variance guess given by

$$\hat{\sigma} = \frac{\text{Median}\,|Y_{ij}|}{0.6745}$$

where Y_{ij} corresponds to all the coefficients of the highest scale subband of the decomposition (where most of the noise is assumed to be present). Alternatively, the noise variance can be obtained in an independent way, for example from the AFM signal variance while not scanning. For the highest frequency subband (universal thresholding) or for each subband (for scale

adaptive thresholding) or for each pixel neighbourhood within subband (for scale and space adaptive thresholding) the variance is computed as

$$\hat{\sigma}_Y^2 = \frac{1}{n^2} \sum_{i,j=1}^{n} Y_{ij}^2$$

Threshold value is finally computed as

$$T(\hat{\sigma}_X) = \hat{\sigma}^2 / \hat{\sigma}_X$$

where

$$\hat{\sigma}_X = \sqrt{\max(\hat{\sigma}_Y^2 - \hat{\sigma}^2, 0)}$$

When threshold for given scale is known, we can remove all the coefficients smaller than threshold value (hard thresholding) or we can lower the absolute value of these coefficients by threshold value (soft thresholding).

DWT denoising can be accessed with *Data Process* → *Integral Transforms* → *DWT Denoise*.

Continuous Wavelet Transform

Continuous wavelet transform (CWT) is an implementation of the wavelet transform using arbitrary scales and almost arbitrary wavelets. The wavelets used are not orthogonal and the data obtained by this transform are highly correlated. For the discrete time series we can use this transform as well, with the limitation that the smallest wavelet translations must be equal to the data sampling. This is sometimes called Discrete Time Continuous Wavelet Transform (DT-CWT) and it is the most used way of computing CWT in real applications.

In principle the continuous wavelet transform works by using directly the definition of the wavelet transform, i.e. we are computing a convolution of the signal with the scaled wavelet. For each scale we obtain by this way an array of the same length N as the signal has. By using M arbitrarily chosen scales we obtain a field $N \times M$ that represents the time-frequency plane directly. The algorithm used for this computation can be based on a direct convolution or on a convolution by means of multiplication in Fourier space (this is sometimes called Fast Wavelet Transform).

The choice of the wavelet that is used for time-frequency decomposition is the most important thing. By this choice we can influence the time and frequency resolution of the result. We cannot change the main features of WT by this way (low frequencies have good frequency and bad time resolution; high frequencies have good time and bad frequency resolution), but we can somehow increase the total frequency of total time resolution. This is directly proportional to the width of the used wavelet in real and Fourier space. If we use the Morlet wavelet for example (real part – damped cosine function) we can expect high frequency resolution as such a wavelet is very well localized in frequencies. In contrary, using Derivative of Gaussian (DOG) wavelet will result in good time localization, but poor one in frequencies.

CWT is implemented in the CWT module that can be accessed with *Data Process* → *Integral Transforms* → *CWT*.

References

[1] Adhemar Bultheel: Learning to swim in a sea of wavelets. Bull. Belg. Math. Soc. Simon Stevin 2 (1995), 1-45, doi:10.36045/bbms/1103408773

[2] S. G. Chang, B. Yu, M. Vetterli: Adaptive wavelet thresholding for image denoising and compression. IEEE Trans. Image Processing 9 (2000) 1532–1536, doi:10.1109/83.862633

[3] S. G. Chang, B. Yu, M. Vetterli: Spatially adaptive wavelet thresholding with context modeling for image denoising. IEEE Trans. Image Processing 9 (2000) 1522–1531, doi:10.1109/83.862630

4.16 Fractal Analysis

In practice objects exhibiting random properties are encountered. It is often assumed that these objects exhibit the self-affine properties in a certain range of scales. Self-affinity is a generalization of self-similarity which is the basic property of most of the deterministic fractals. A part of self-affine object is similar to whole object after anisotropic scaling. Many randomly rough surfaces are assumed to belong to the random objects that exhibit the self-affine properties and they are treated self-affine statistical fractals. Of course, these surfaces can be studied using atomic force microscopy (AFM). The results of the fractal analysis of the self-affine random surfaces using AFM are often used to classify these surfaces prepared by various technological procedures [1,2,3,4].

Within Gwyddion, there are different methods of fractal analysis implemented within *Data Process* → *Statistics* → *Fractal analysis*.

Cube counting method [1,2] is derived directly from a definition of box-counting fractal dimension. The algorithm is based on the following steps: a cubic lattice with lattice constant l is superimposed on the z-expanded surface. Initially l is set at $X/2$ (where X is length of edge of the surface), resulting in a lattice of $2 \times 2 \times 2 = 8$ cubes. Then $N(l)$ is the number of all cubes that contain at least one pixel of the image. The lattice constant l is then reduced stepwise by factor of 2 and the process repeated until l equals to the distance between two adjacent pixels. The slope of a plot of $\log N(l)$ versus $\log 1/l$ gives the fractal dimension D_f directly.

Triangulation method [1] is very similar to cube counting method and is also based directly on the box-counting fractal dimension definition. The method works as follows: a grid of unit dimension l is placed on the surface. This defines the location of the vertices of a number of triangles. When, for example, $l = X/4$, the surface is covered by 32 triangles of different areas inclined at various angles with respect to the xy plane. The areas of all triangles are calculated and summed to obtain an approximation of the surface area $S(l)$ corresponding to l. The grid size is then decreased by successive factor of 2, as before, and the process continues until l corresponds to distance between two adjacent pixel points. The slope of a plot of $S(l)$ versus $\log 1/l$ then corresponds to $D_\mathrm{f} - 2$.

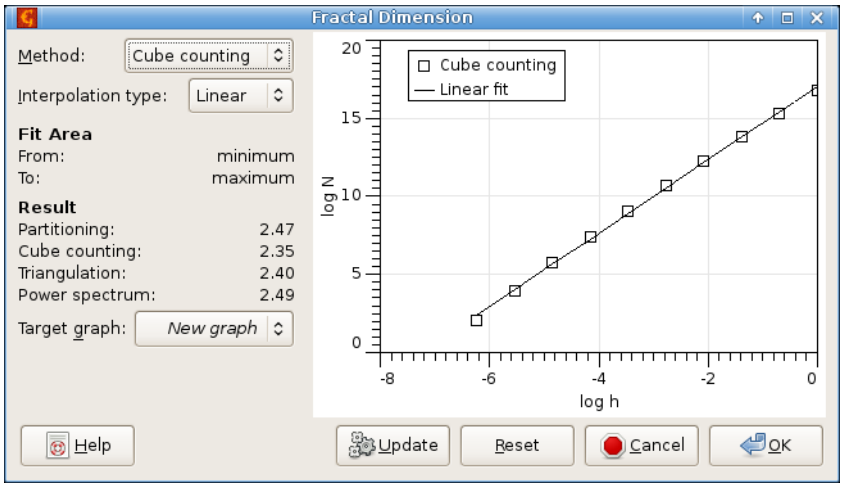
Variance (partitioning) method [3,4] is based on the scale dependence of the variance of fractional Brownian motion. In practice, in the variance method one divides the full surface into equal-sized squared boxes, and the variance (power of RMS value of heights), is calculated for a particular box size. The fractal dimension is evaluated from the slope β of a least-square regression line fit to the data points in log-log plot of variance as $D_\mathrm{f} = 3 - \beta/2$.

Power spectrum method [3,4,5] is based on the power spectrum dependence of fractional Brownian motion. In the power spectrum method, every line height profiles that forms the image is Fourier transformed and the power spectrum evaluated and then all these power spectra are averaged. The factal dimension is evaluated from the slope β of a least-square regression line fit to the data points in log-log plot of power spectrum as $D_\mathrm{f} = 7/2 + \beta/2$.

Structure function (HHCF) method is similar to the power spectrum method, but instead of the power spectrum, the structure function (height-height correlation function) is fitted around zero. The fractal dimension is evaluated from the slope β of a least-square regression line fit to the data points in log-log plot of variance as $D_\mathrm{f} = 3 - \beta/2$.

The axes in *Fractal Dimension* graphs always show already logarithmed quantities, therefore the linear dependencies mentioned above correspond to straight lines there. Frequently only a part of the plotted curve is a straight line as the end tend to be influenced by various systematic errors and artefacts. The measure of the axes should be treated as arbitrary.

Note that that results of different methods differ. This fact is caused by systematic error of different fractal analysis approaches.



Fractal Dimension dialog.

Moreover, the results of the fractal analysis can be influenced strongly by the tip convolution. We recommend therefore to check the certainty map before fractal analysis. In cases when the surface is influenced a lot by tip imaging, the results of the fractal analysis can be misrepresented strongly.

Note, that algorithms that can be used within the fractal analysis module are also used in Fractal Correction module and Fractal Correction option of Remove Spots tool.

References

[1] C. Douketis, Z. Wang, T. L. Haslett, M. Moskovits: Fractal character of cold-deposited silver films determined by low-temperature scanning tunneling microscopy. Physical Review B 51 (1995) 11022, doi:10.1103/PhysRevB.51.11022

[2] W. Zahn, A. Zösch: The dependence of fractal dimension on measuring conditions of scanning probe microscopy. Fresenius J Analen Chem 365 (1999) 168-172, doi:10.1007/s002160051466

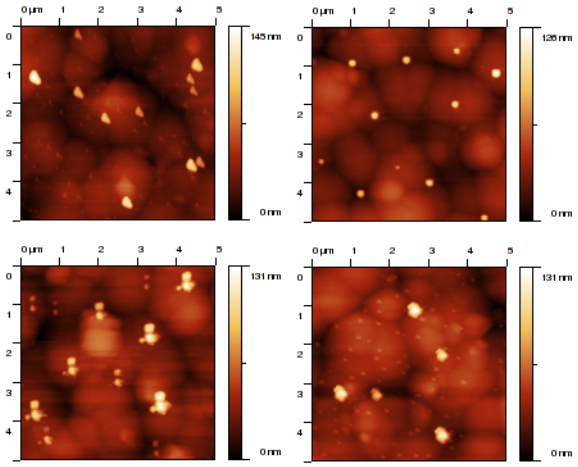
[3] A. Van Put, A. Vertes, D. Wegrzynek, B. Treiger, R. Van Grieken: Quantitative characterization of individual particle surfaces by fractal analysis of scanning electron microscope images. Fresenius J Analen Chem 350 (1994) 440-447, doi:10.1007/-BF00321787

[4] A. Mannelquist, N. Almquist, S. Fredriksson: Influence of tip geometry on fractal analysis of atomic force microscopy images. Appl. Phys. A 66 (1998) 891-895, doi:10.1007/s003390051262

[5] W. Zahn, A. Zösch: Characterization of thin film surfaces by fractal geometry. Fresenius J Anal Chem 358 (1997) 119-121, doi:10.1007/s002160050360

4.17 Tip Convolution Artefacts

Tip convolution artefact is one of the most important error sources in SPM. As the SPM tip is never ideal (like delta function) we often observe a certain degree of image distortion due to this effect. We can even see some SPM tips imaged on the surface scan while sharp features are present on the surface.



Images of ZnSe surface measured with four different SPM tips (more or less broken ones).

We can fortunately simulate and/or correct the tip effects using algorithms of dilation and/or erosion, respectively. These algorithms were published by Villarubia (see [1]).

Obtaining the Tip Geometry

For studying the tip influence on the data we need to know tip geometry first. In general, the geometry of the SPM tip can be determined in several ways:

1. Use manufacturer's specifications (tip geometry, apex radius and angle).
2. Use scanning electron microscope of other independent technique to determine tip properties.
3. Use known tip characterizer sample (with steep edges).
4. Use blind tip estimation algorithm together with tip characterizers or other suitable samples.

Tip Modelling

Data Process → Tip and Indentation → Model Tip

Tip modelling implements the first approach. Using tip modelling most of the tips with simple geometries can be simulated. This way of tip geometry specification can be very efficient namely when we need to check only certainty map of perform tip convolution simulation.

The function offers several usual model shapes such as pyramids, cones or paraboloids. Each is described by a subset of the following parameters:

Number of sides The number of pyramid sides applies to pyramidal tips (which do not have a fixed number of sides by definition).

Tip slope Angle of the pyramid side, measured from the vertical plane. Small angles mean sharp tips and large angles means shallow tips.

Tip rotation Rotation with respect to a base orientation, measured anti-clockwise. It applies to tips which are not rotationally symmetric.

Tip apex radius Mean radius of curvature at the tip apex (applies to all tip shapes, except the delta-function). If the tip is not anisotropic, this is simply the radius of curvature in all directions.

Tip anisotropy Ratio expressing how much the tip is squashed along one of the two orthogonal directions with respect to the other. Only the elliptic parabola model has an anisotropy parameter.

The tip image dimensions are determined automatically using the source data to cover the data z-range.

Blind Estimation of Tip Shape

Data Process → Tip and Indentation → Blind Estimation

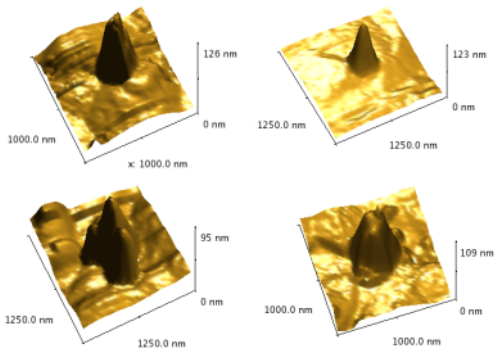
To obtain more detailed (and more realistic) tip structure blind tip estimation algorithm can be used. Blind tip estimation algorithm is an extension of the well-known fact that on some surface data we can see images of certain parts of tip directly. The algorithm iterates over all the surface data and at each point tries to refine each tip point according to steepest slope in the direction between concrete tip point and tip apex.

We can use two modification of this algorithm within Gwyddion: *partial* tip estimation that uses only limited number of highest points on the image and *full* tip estimation that uses full image (and is much slower therefore). Within Gwyddion tip estimation module we can use also partial tip estimation results as starting point for full estimation. This improves the speed of full tip estimation.

The typical procedure is

1. Use *Reset Tip* to start afresh, for instance after changing parameters.
2. Run the partial estimation using *Run Partial*.
3. Run the full estimation using *Run Full*.

It is possible – and often useful – to run the full estimate with several topographical images in sequence. This can be achieved by selecting successively the images as *Related data* and running the estimation.



SPM tips obtained from data of previous figure using blind estimation algorithm.

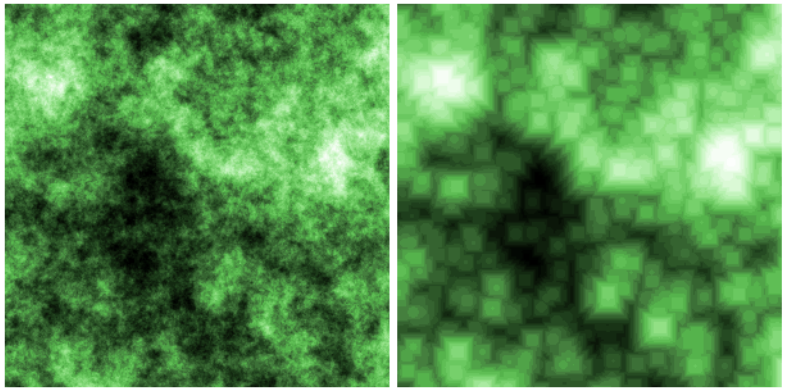
The tip shape can change during the scanning. Therefore, it is sometimes useful to inspect how its blind estimation evolves with the scan lines. Of course, estimation from a single scan line is impossible. Hence, the image is split to horizontal stripes and the tip is estimated for each stripe.

The stripe-wise processing is enabled by *Split to stripes*, which also controls the number of stripes. You can choose which tip is displayed using *Preview stripe*. A tip radius graph is displayed below the preview – and can be created as output when *Plot size graph* is enabled. The function can also produce the full sequence of estimated tip images, which is enabled by *Create tip images*.

Tip Convolution and Surface Reconstruction

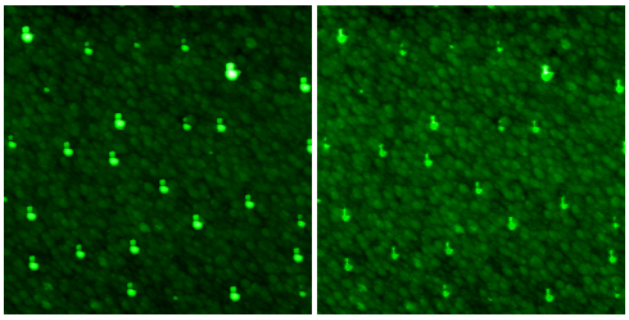
When we know tip geometry, we can use tip convolution (dilation) algorithm to simulate data acquisition process. For doing this use Dilation module (*Data Process* → *Tip and Indentation* → *Dilation*). This can be in particular useful when working with data being result of some numerical modelling (see e.g. [2]).

Note this algorithm (as well as the following two) requires compatible scan and tip data, i.e. the physical dimensions of a scan pixel and of a tip image pixels have to be equal. This relation is automatically guaranteed for tips obtained by blind estimate when used on the same data (or data with an identical measure). If you obtained the tip image by other means, you may need to resample it.



Simulated fractal surface before (left) and after (right) tip convolution.

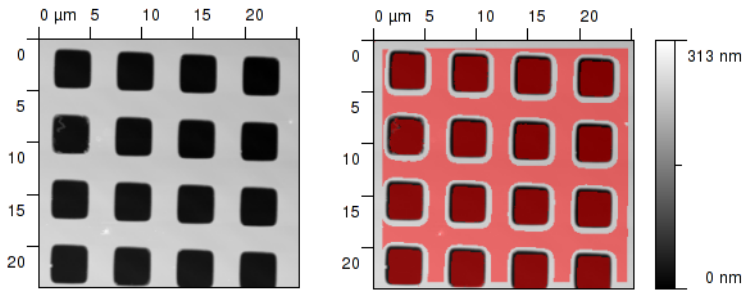
The opposite of the tip convolution is surface reconstruction (erosion) that can be used to correct partially the tip influence on image data. For doing this, use Surface Reconstruction function (*Data Process* → *Tip and Indentation* → *Surface Reconstruction*). Of course, the data corresponding to points in image not touched by tip (e. g. pores) cannot be reconstructed as there is no information about these points.



Original and reconstructed image of ZnSe imaged by broken SPM tip.

Certainty Map

As it can be seen, the most problematic parts of SPM image are data points where tip did not touch the surface in a single point, but in multiple points. There is a loss of information in these points. Certainty map algorithm can mark points where surface was probably touched in a single point.



Certainty map obtained from standard grating. Note that the modelled tip parameters were taken from datasheet here for illustration purposes. (left) – sample, (right) – sample with marked certainty map.

Certainty map algorithm can be therefore used to mark data in the SPM image that are corrupted by tip convolution in an irreversible way. For SPM data analysis on surfaces with large slopes it is important to check always presence of these points. Within Gwyddion you can use Certainty Map function for creating these maps (*Data Process* → *Tip and Indentation* → *Certainty Map*).

References

[1] J. S. Villarrubia: Algorithms for Scanned Probe Microscope Image Simulation, Surface Reconstruction, and Tip Estimation. J. Res. Natl. Inst. Stand. Technol. 102 (1997) 425, doi:10.6028/jres.102.030

[2] P. Klapetek, I. Ohlídal: Theoretical analysis of the atomic force microscopy characterization of columnar thin films. Ultramicroscopy 94 (2003) 19–29, doi:10.1016/S0304-3991(02)00159-6

4.18 Force and Indentation

Analyze Imprint 

Data Process → *SPM Modes* → *Force and Indentation* → *Analyze Imprint...*

The module identifies different characteristic parts of an indent and calculates the corresponding projected and developed areas (A_p and A_d) and volumes.

Impression Is defined as the region around the local minimum with the same normal vector (up to symmetry). This corresponds to the usual definition of the contact area.

Inner pile-up Part of the impression lying above the reference plane.

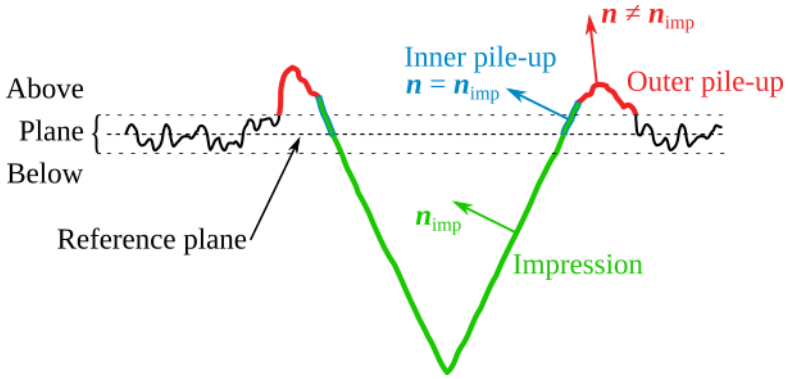
Outer pile-up Part of the image above the reference plane but with a different normal vector than the impression.

Special points Include the maximum, minimum and points along steepest slope profiles.

Expected projected and developed areas Refer to the values obtained by evaluating the theoretical area function of the chosen indentor type at the depth of the imprint.

Volume above-below Volume above-below is the difference between the volume above the reference plane and the volume below.

Note, that the module levels the data by fitting planes along the borders of the image. This may lead to mismatch between some of the values returned by this model and analoguous ones returned by the Statistical Quantities tool. The data should be leveled manually in order to reduce this effect.



Scheme of different regions of an indentation: above, below, plane, impression, inner pile-up and outer pile-up.

4.19 Convolution and deconvolution

Several Gwyddion functions do convolutions and deconvolutions of some sort. A subtle point which is occasionaly a source of confusion is the difference between continuous and discrete operations. When images f and g are physical quantities, i.e. fields sampled in space, their convolution h

$$h(x) = \int_A f(x-u)\, g(u)\, \mathrm{d}u$$

is again a physical field. Its physical dimension (units) is

$$\dim(h) = \dim(f)\, \dim(g)\, \dim(A)$$

For images $\dim(A)$ has the dimension of image area, i.e. usually square metres.

When the integral is realized using a discrete Riemann sum

$$h_n \approx \Delta A \sum_k f_{n-k} g_k$$

the area appears as the area of one pixel ΔA. This is how all Gwyddion convolution, deconvolution and transfer function operations work by default, i.e. with the *Normalize as integral* option enabled.

In digital image processing physical units are sometimes ignored and convolution understood as simple discrete convolution of images

$$h_n = \sum_k f_{n-k} g_k$$

It is possible to obtain this behaviour by disabling the option *Normalize as integral* in a particular function. You should be aware the result of the operation is then no longer a physical quantity sampled in space (it depends on pixel size) and the units are different. The only function which works this way by default is Convolution Filter, where one of the operands is given as a small matrix of unitless numbers.

Image convolution 

Data Process → *Multidata* → *Convolve*

Convolution of two images can be performed using Convolve (see Convolution Filter for convolution with a small kernel, entered numerically). The module has only a few options:

Convolution kernel The smaller image to convolve the larger image with, for instance a tip transfer function image. The dimensions of one pixel in both images must correspond.

Exterior type What values should be assumed for pixels outside the image (as in Extend).

Output size The size of the resulting image with respect to which pixels entered the calculation. *Crop to interior* cuts the image to consists only of pixels that are sufficiently far from the boundary that the exterior does not matter. *Keep size* creates an image with the same as the input image. Finally, *Extend to convolved* means the resulting image will be extended on each side by half the kernel size.

Image deconvolution 

Data Process → Multidata → Deconvolve

A simple regularized deconvolution of two images can be performed using Deconvolve. L-curve method can be used to search for the optimum regularization parameter, however its applicability highly depends on the data. The module has the following options:

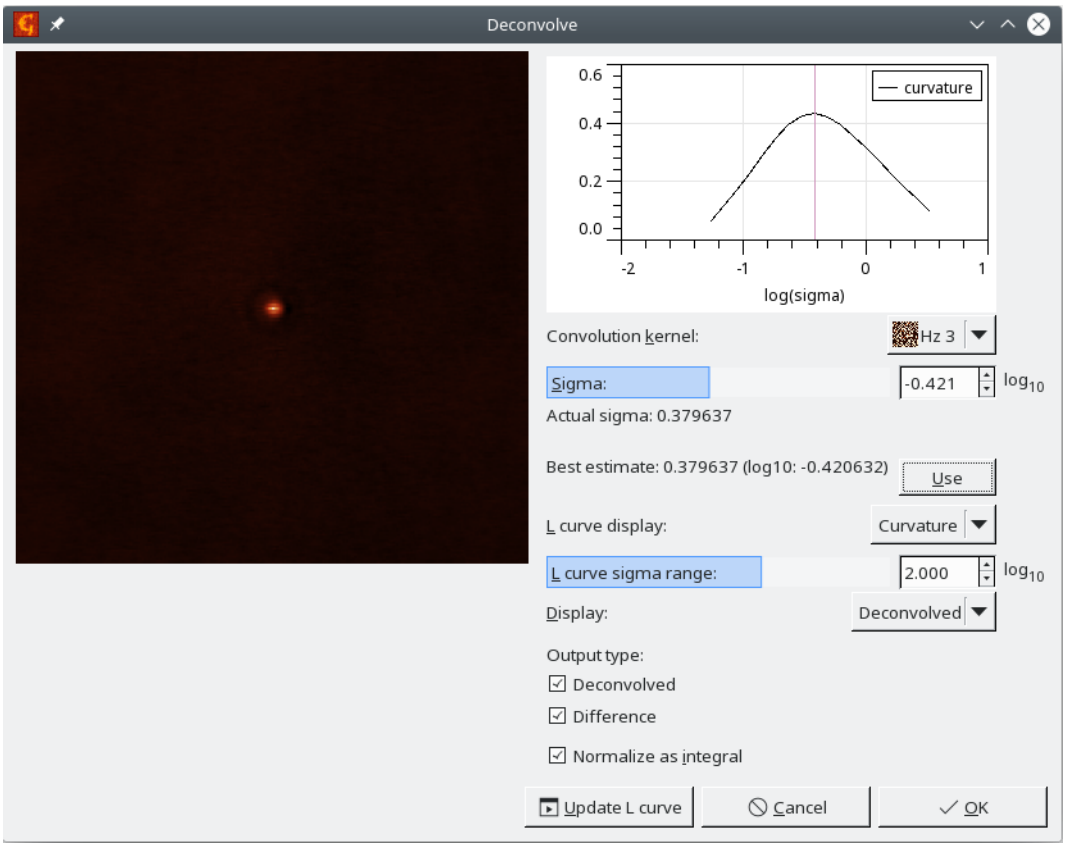
Convolution kernel The kernel to be deconvolved from the current image.

Sigma Regularization parameter.

L-curve display The data to be shown in the optional L-curve plot, when this is computed. The L-curve consist of two data sources: difference between the result convolved with the kernel and the original data and norm of the result. These two are plotted in a log-log graph providing a L-curve, but their dependences on the regularization parameters can be plotted also individually. The curvature of the resulting curve vs. regularization parameter gives the usual criterion for the best regularization parameters, seeking for the curve maximum.

L-curve sigma range Range of the regularization parameter to be used in the L-curve calculation. This is, similarly to the regularization parameter, logarithmic, so it can span very large range, not necessarily suitable for the particular data set.

Output type *Deconvolved* produces the deconvolved image; *Difference* calculates the difference between reconvolved image and the input.



Screenshot of the deconvolve dialogue after L-curve data were computed.

Transfer function estimation

The *probe transfer function* (or point spread function) can be estimated from measured data, even noisy, when the corresponding ideally sharp signal is available. Gwyddion currently implements a few relatively simple methods.

Data Processing → Statistics → Transfer Function Guess. The user interface of the module is and a typical performance on simulated noisy data is shown in the following figures.

The sharp signal is selected as *Ideal response*. It must have the same dimensions as the image. In contrast to the general Deconvolve module this module is optimized for the statistical properties of the transfer functions. There are currently three available deconvolution methods:

Regularised filter Simple frequency-space filter with constant regularisation term. It produces transfer function image of the same size as the input images. Option *TF zoom* can be used to display zoomed central part of the image.

Least squares The transfer function is found by solving the least squares problem corresponding to the deconvolution. The size of transfer function can be chosen arbitrarily. Usually it is much smaller than the input images.

Parameter *Size* controls the transfer function image size. If *Border* is nonzero, the transfer function image is enlarged by this value before solving the least squares problem and then the border is cut. A small border of 2 or 3 often improves the result because errors tend to accumulate in the edge pixels. Both sizes can be estimated automatically using *Estimate Size*.

Wiener filter Pseudo-Wiener filter assuming the spectral density of the output can be estimated by the spectral density of the input image. It is again a frequency-domain filter, producing transfer function image of the same size as the input images. Option *TF zoom* can be used to display zoomed central part of the image.

Two important parameters influence the result. The dimensionless regularisation parameter σ, which is used in all the methods, but is not directly comparable between them. It can be chosen manually. Since suitable values can differ by several orders of magnitude, it is presented as base-10 logarithm. However, the module can find a suitable value also automatically when *Fit Sigma* is pressed.

Windowing type determines the windowing function applied to both input images before deconvolution. Unless the input data are periodic, windowing is a necessity to avoid cross-like artefacts in the transfer function. The default Welch window is usually a good choice.

A few characteristics of the transfer function are immediately displayed in the bottom part of the dialogue as *Result*:

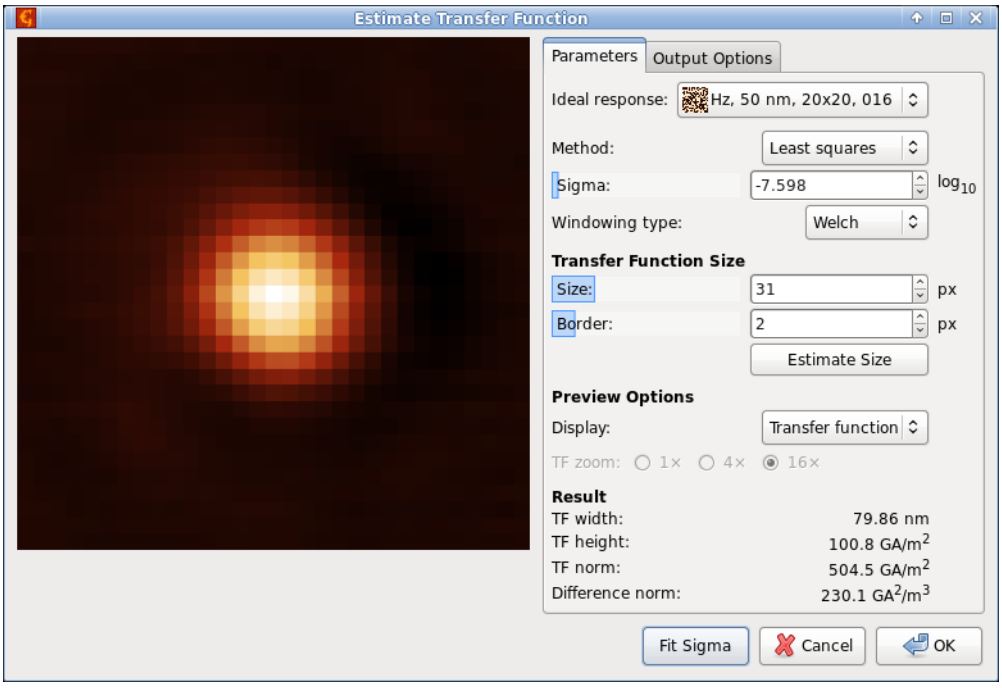
TF width Width, calculated as square root of dispersion when the transfer function is taken as a distribution. Only the central peak is included in the calculation (which is important namely for regularised and Wiener filters). Usually, the optimal reconstruction has also small width.

TF height Maximum value of the transfer function. When it starts to decrease noticeably, the regularisation parameter is probably too high.

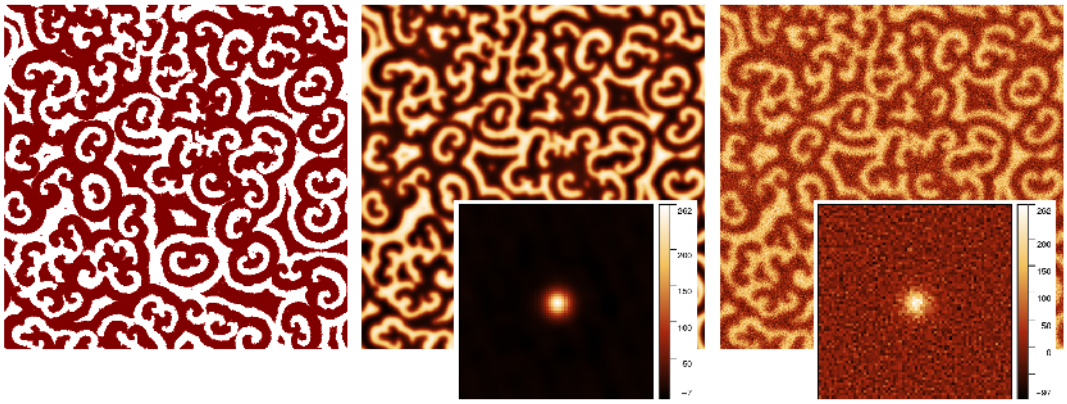
TF norm Mean square value of the transfer function.

Difference norm Mean square value of the difference between the input image and ideal sharp image convolved with the reconstructed transfer function.

The tab *Output Options* allows choosing the images the function will produce: the transfer function (*Transfer function*), its convolution with the ideal sharp image (*Convolved*) and/or the difference between the input and the convolved image (*Difference*). For methods which produce full-size transfer function images, *Only zoomed part* limits the output image size to the zoomed part displayed in the dialogue.



Screenshot of the transfer function estimation module.



Sample results of the transfer function estimation module.

An alternative method of TF estimation is the fitting of an explicit function form, for instance Gaussian. The free parameters of the function are sought to minimise the sum of squared differences between measured data and ideal response convolved with the TF. Since the minimisation can be performed entirely in the frequency domain and does not require repeated convolution, this method is simple and fast, albeit somewhat crude. It is available as *Data Processing* → *Statistics* → *Transfer Function Fit* with three model functions currently implemented: fully symmetrical Gaussian, anisotropic Gaussian which can have different widths in the horizontal and vertical directions, and TF which is exponential in the frequency space.

4.20 Multiple Data

Arithmetic

Data Process → *Multidata* → *Arithmetic*

Data Arithmetic module enables to perform arbitrary point-wise operations on a single image or on the corresponding points of several images (currently up to eight). And although it is not its primary function it can be also used as a calculator with immediate expression evaluation if a plain numerical expression is entered. The expression syntax is described in section Expressions.

The expression can contain the following variables representing values from the individual input images:

Variable	Description
`d1`, ..., `d8`	Data value at the pixel. The value is in base physical units, e.g. for height of 233 nm, the value of `d1` is 2.33e-7.
`m1`, ..., `m8`	Mask value at the pixel. The mask value is either 0 (for unmasked pixels) or 1 (for masked pixels). The mask variables can be used also if no mask is present; the value is then 0 for all pixels.
`bx1`, ..., `bx8`	Horizontal derivative at the pixel. Again, the value is in physical units. The derivative is calculated as standard symmetrical derivative, except in edge pixels where one-side derivative is taken.
`by1`, ..., `by8`	Vertical derivative at the pixel, defined similarly to the horizontal derivative.
`x`	Horizontal coordinate of the pixel (in real units). It is the same in all images due to the compatibility requirement (see below).
`y`	Vertical coordinate of the pixel (in real units). It is the same in all images due to the compatibility requirement (see below).

In addition, the constant π is available and can be typed either as π or **`pi`**.

All images that appear in the expression have to be compatible. This means their dimensions (both pixel and physical) have to be identical. Other images, i.e. those not actually entering the expression, are irrelevant. The result is always put into a newly created image in the current file (which may be different from the files of all operands).

Since the evaluator does not automatically infer the correct physical units of the result the units have to be explicitly specified. This can be done by two means: either by selecting an image that has the same value units as the result should have, or by choosing option *Specify units* and typing the units manually.

The following table lists several simple expression examples:

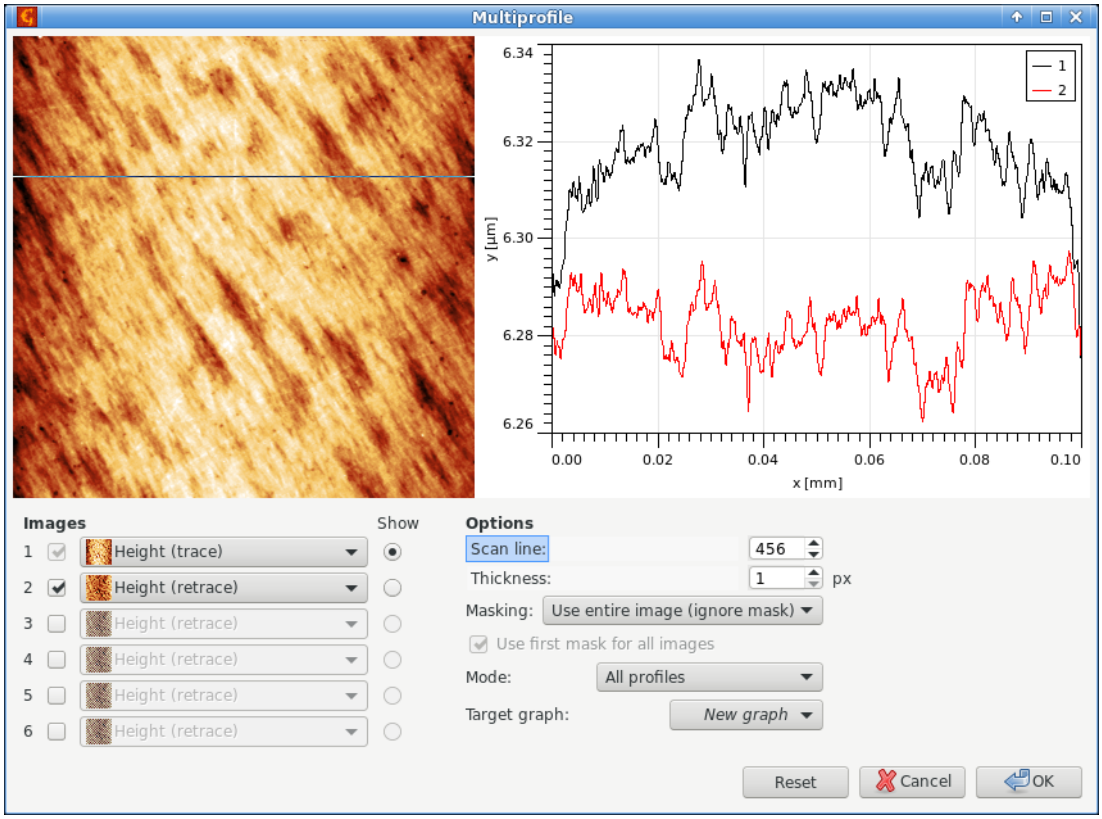
Expression	Meaning
`-d1`	Value inversion. The result is very similar to Invert Value, except that Invert Value reflects about the mean value while here we simply change all values to negative.
`(d1 - d2)^2`	Squared difference between two images.
`d1 + m1*1e-8`	Modification of values under mask. Specifically, the value 10^{-8} is added to all masked pixels.
`d1*m3 + d2*(1-m3)`	Combination of two images. Pixels are taken either from image 1 or 2, depending on the mask on image 3.

In the calculator mode the expression is immediately evaluated as it is typed and the result is displayed below Expression entry. No special action is necessary to switch between image expressions and calculator: expressions containing only numeric quantities are immediately evaluated, expressions referring to images are used to calculate a new image. The preview showing the result of an operation with images is not immediately updated as you type; you can update it either by clicking *Update* or just pressing **Enter** in the expression entry.

Profiles from multiple images

Data Process → *Multidata* → *Multiprofile*

It is often useful to compare the corresponding scan lines from different images – in particular trace and retrace. This can be done using Multiprofile and selecting the two images as 1 and 2 in the *Images* list. The selected scan line is displayed on the image. It can be moved around with mouse or the scan line number can be entered numerically using *Scan line*.



Basic usage of multiprofile module, with trace and retrace images selected as 1 and 2 and a the corresponding pair of profiles displayed in the graph.

The profiles can be read simultaneously from up to six images. Enable and disable individual images using the checkboxes next to the image number. Any of the enabled images can be displayed in the small preview, which is controlled by selecting them using *Show*.

Since the graph shows mutually corresponding profiles, all images must have the same pixel and physical dimensions and must also represent the same physical quantity. Image 1 is special in this regard. It cannot be unselected and it defines the dimensions and other properties. In other words, all the other images must be compatible with image 1.

Options in the lower right part include the standard thickness (averaging) control, the same as for the normal profile tool, masking options and target graph for the extracted profiles. Masks can be either taken from the corresponding images, or the mask on the first image can be used for all – if *Use first mask for all images* is ticked.

Instead of extracting the individual profiles, the module can also do simple statistics and plot summary curves – mean curve with lower and upper bounds. This is controlled by *Mode*.

Detail Immersion 

Data Process → Multidata → Immerse

Immerse insets a detailed, high-resolution image into a larger image. The image the function was run on forms the large, base image.

The detail can be positioned manually on the large image with mouse. Button *Improve* can then be used to find the exact coordinates in the neighbourhood of the current position that give the maximum correlation between the detail and the large image. Or the best-match position can be searched through the whole image with *Locate*.

It should be noted that correlation search is insensitive to value scales and offsets, therefore the automated matching is based solely on data features, absolute heights play no role.

Result Sampling controls the size and resolution of the result image:

Upsample large image The resolution of the result is determined by the resolution of the inset detail. Therefore the large image is scaled up.

Downsample detail The resolution of the result is determined by the resolution of the large image. The detail is downsampled.

Detail Leveling selects the transform of the z values of the detail:

None No z value adjustment is performed.

Mean value All values of the detail image are shifted by a constant to make its mean value match the mean value of the corresponding area of the large image.

Merging 

Data Process → Multidata → Merge

Images that form parts of a larger image can be merged together with Merge. The image the function was run on corresponds to the base image, the image selected with *Merge with* represents the second operand. The side of the base image the second one will be attached to is controlled with *Put second operand* selector.

If the images match perfectly, they can be simply placed side by side with no adjustments. This behaviour is selected by option *None* of alignment control *Align second operand*.

However, usually adjustments are necessary. If the images are of the same size and aligned in direction pependicular to the merging direction the only degree of freedom is possible overlap. The *Join* aligment method can be used in this case. Unlike in the correlation search described below, the absolute data values are matched. This makes this option suitable for merging even very slowly varying images provided their absolute height values are well defined.

Option *Correlation* selects automated alignment by correlation-based search of the best match. The search is performed both in the direction parallel to the attaching side and in the perpendicular direction. If a parallel shift is present, the result is expanded to contain both images fully (with undefined data filled with a background value).

Option *Boundary treatment* is useful only for the latter case of imperfectly aligned images. It controls the treatment of overlapping areas in the source images:

First operand Values in overlapping areas are taken from the first, base image.

Second operand Values in overlapping areas are taken from the second image.

Smooth A smooth transition between the first and the second image is made through the overlapping area by using a weighted average with a suitable weighting function.

Stitching

Data Process → *Multidata* → *Stitch*

Stitching is an alternative to the merge module described above. It is mainly useful when the relative positions of the image parts are known exactly because the positions entered numerically. They are initialised using image offsets, so if these are correct, the stitched image is formed automatically. Buttons *Restore* for each part revert manually modified offsets to the initial ones.

Mutual Crop

Data Process → *Multidata* → *Mutual Crop*

Two slightly different images of the same area (for example, before and after some treatment) can be croped to intersecting area (or non-intersecting parts can be removed) with this module.

Intersecting part is determined by correlation of larger image with center area of smaller image. Images resolution (pixels per linear unit) should be equal.

The only parameter now is *Select second operand* - correlation between it and current image will be calculated and both images will be cropped to remove non-intersecting near-border parts.

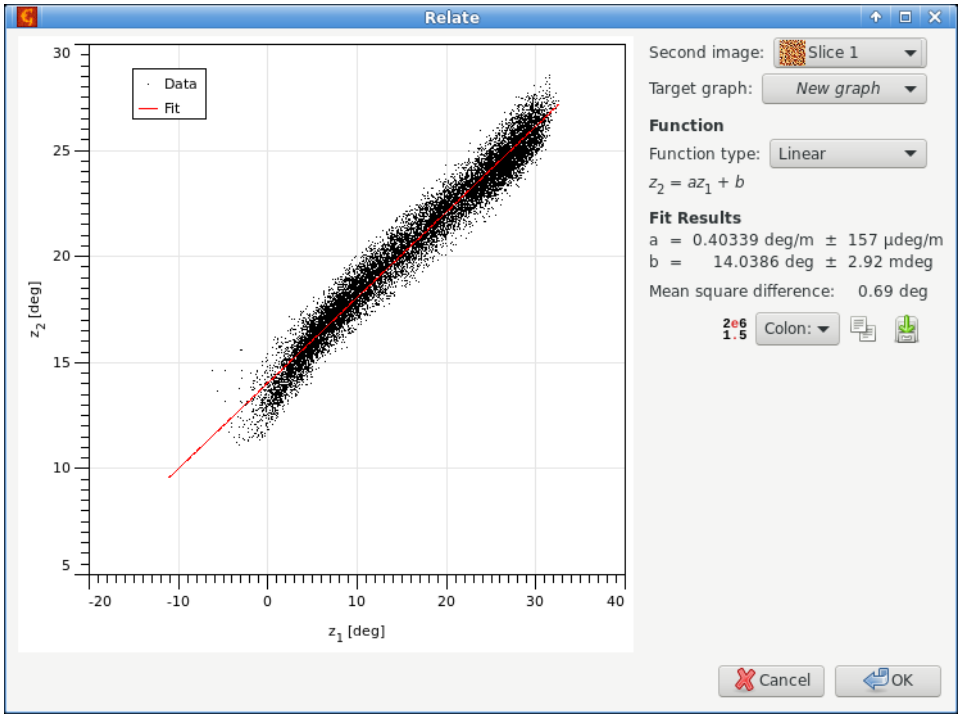
Relation

Data Process → *Multidata* → *Relate*

When data in two images are related in a simple manner, for instance they differ by a constant calibration factor, this function helps finding the relation.

The graph in the left part shows pixel values of the second image (selected as *Second image*) plotted as a function of corresponding pixel values in the current image. When a simple relation between the values exist, all the points lie on a single curve. Note that for large images only a small, randomly selected subset of pixel values is plotted in the graph.

Several simple relations can be fitted to the point data, such as proportion, offset, linear relation or quadratic relation. The fit parameters are immediately evaluated and displayed in the table below and the corresponding relation plotted. All pixel values are used for fitting, even if only a subset is displayed.



An almost-linear relation between two slices of volume MFM data at different heights, as displayed and fitted by the relation module.

Cross-Correlation

Data Process → Multidata → Cross-correlation

This module finds local correlations between details on two different images. As an ideal output, the shift of every pixel on the first image as seen on the second image is returned. This can be used for determining local changes on the surface while imaged twice (shifts can be for example due to some sample deformation or microscope malfunction).

For every pixel on the first operand (actual window), the module takes its neighbourhood and searches for the best correlation in the second operand within defined area. The position of the correlation maximum is used to set up the value of shift for the mentioned pixel on the first operand.

Second operand Image to be used for comparison with the first operand - base image.

Search size Used to set the area whera algorithm will search for the local neighbourhood (on the second operand). Should be larger than window size. Increase this size if there are big differences between the compared images.

Window size Used to set the local neighbourhood size (on the first operand). Should be smaller than search size. Increasing this value can improve the module functionality, but it will surely slow down the computation.

Output type Determines the output (pixel shift) format.

Add low score threshold mask For some pixels (with not very pronounced neighbourhood) the correlation scores can be small everywhere, but the algorithm anyway picks some maximum value from the scores. To see these pixels and possibly remove them from any further considerations you can let the module to set mask of low-score pixel shifts that have larger probability to be not accurately determined.

Correlation Search 

Data Process → Multidata → Correlation Search

This module searches for a given detail within the image base. It can produce a correlation score image or mark resulting detail position using a mask on the base image.

Correlation kernel Detail image to be found on the base image. The dimensions of one pixel in both images must correspond.

Use mask If the detail image has a mask you can enable this option. Only the pixels covered by the mask then influence the search. This can help with marking of details that have irregular shape.

Output type There are several possibilities what to output: local correlation maxima (single points), masks of kernel size for each correlation maximum (good for presentation purposes), or simply the correlation score.

Correlation method Several correlation score computation methods are available. There are two basic classes. For *Correlation* the score is obtained by integrating the product of the kernel and base images. For *Height difference* the score is obtained by integrating the squared difference between the kernel and base images (and then inverting the sign to make higher scores better). For both methods the image region compared to the kernel can be used as-is (raw), it can be levelled to the same mean value or it can be levelled to the same mean value and normalised to the same sum of squares, producing the score.

Threshold Threshold for determining whether the local maximum will be treated as “detail found here”.

Regularization When the base image regions are locally normalised, very flat regions can lead to random matches with the detail image and other odd results. By increasing the regularization parameter flat regions are suppressed.

Neural network processing

Neural network processing can be used to calculate one kind of data from another even if the formula or relation between them is not explicitly known. The relation is built into the network implicitly by a process called training which employs pairs of known input and output data, usually called model and signal. In this process, the network is optimised to reproduce as well as possible the signal from model. A trained network can then be used to process model data for which the output signal is not available and obtain – usually somewhat approximately – what the signal would look like. Another possible application is the approximation of data processing methods that are exact but very time-consuming. In this case the signal is the output of the exact method and the network is trained to reproduce that.

Since training and application are two disparate steps they are present as two different functions in Gwyddion.

Training 

Data Process → *Multidata* → *Neural Network Training*

The main functions that control the training process are contained in tab *Training*:

Model Model data, i.e. input for training. Multiple models can be chosen sequentially for training (with corresponding signals).

Signal Signal data for training, i.e. the output the trained network should produce. The signal image must be compatible with model image, i.e. it must have the same pixel and physical dimensions.

Training steps Number of training steps to perform when *Train* is pressed. Each step consists of one pass through the entire signal data. It is possible to set the number of training steps to zero; no training pass is performed then but the model is still evaluated with the network and you can observe the result.

Train Starts training. This is a relatively slow process, especially if the data and/or window size are large.

Reinitialize Reinitializes the neural network to an untrained state. More precisely, this means neuron weights are set to random numbers.

Masking Mode It is possible to train the network only on a subset of the signal, specified by a mask on the signal data. (Masking of model would be meaningless due to the window size.)

Neural network parameters can be modified in tab *Parameters*. Changing either the window dimensions or the number of hidden nodes means the network is reinitialized (as if you pressed *Reinitialize*).

Window width Horizontal size of the window. The input data for the network consists of an area around the model pixel, called window. The window is centered on the pixel so, odd sizes are usually preferred.

Window height Vertical size of the window.

Hidden nodes Number of nodes in the "hidden" layer of the neural network. More nodes can lead to more capable network, on the other hand, it means slower training and application. Typically, this number is small compared to the number of pixels in the window.

Power of source XY The power in which the model lateral dimensions units should appear in the signal. This is only used when the network is applied.

Power of source Z The power in which the model "height" units should appear in the signal. This is only used when the network is applied.

Fixed units Fixed units of the result. They are combined with the other units parameters so if you want result units that are independent of input units you have to set both powers to 0. This is only used when the network is applied.

Trained neural network can be saved, loaded to be retrained on different data, etc. The network list management is similar to raw file presets.

In addition to the networks in the list, there is one more unnamed network and that of the network currently in training. When you load a network the network in training becomes a copy of the loaded network. Training then does not change the named networks; to save the network after training (under existing or new name) you must explicitly use *Store*.

Application 

Data Process → *Multidata* → *Apply Neural Network*

Application of a trained neural network is simple: just choose one from the list and press *OK*. The unnamed network currently in training is also present in the list under the label "In training".

Since neural networks process and produce normalised data, it does not perserve proportionality well, especially if the scale of training model differs considerably from the scale of real inputs. If the output is expected to scale with input you can enable option *Scale proportionally to input* that scales the output with the inverse ratio of actual and training input data ranges.

4.21 Magnetic force microscopy

In MFM measurements we collect information about stray field distribution above sample, typically in one layer in fixed height above sample during one measurement. Together with this signal, we collect also topography, similarily to all the other SPM techniques.

The functions specific to MFM can be found in the *Data Processing* → *SPM Modes* → *Magnetic*. The individual modules and data processing algorithms discussed in the following paragraphs can be split into several categories:

- utility functions, such as conversion of different quantities to force gradient,
- modules for simulation of the MFM response on different samples (e.g. perpendicular media or a thin film stripe with electrical current),
- probe analysis modules based on use of known samples, and
- sets of functions for handling multiple channel data, e.g. data measured with opposite tip magnetisation.

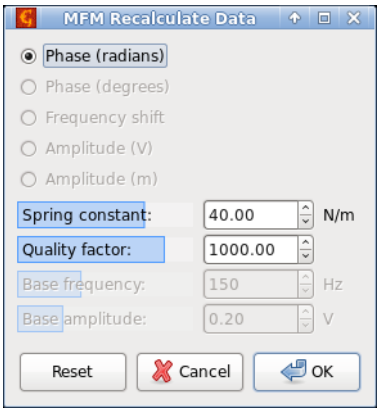
The last category includes general functions such as data arithmetic which can add subtract images and generally combine them using arithmetic expressions or mutual crop which cuts images to the common sub-region (and is often a prerequsite for multi-data operations).

Addition and subtraction of measurements performed with two different probe magnetisation directions was proposed as the means of potential improvement of the process of splitting the magnetic and other contributions [Cambel11]. If measurements with different current direction are also available, we have four possible variants of how the electrostatic and magnetic forces are mutually oriented. All four can be comined to using data arithmetic to supress the noise in the data

Conversion to force gradient

Function *Recalculate to Force Gradient* performs conversion of various measured quantities to force gradient. The input image must correspond to one of the quantities for which the conversion is implemented, phase, frequency shift or amplitude (in Volts or metres). The type of input data is automatically determined and indicated by the input quantity choice in the upper part.

The conversion requires entering instrumental parameters, which differ depending on the input data and may include the sprint constant, quality factor or base amplitude.



Screenshot of MFM data conversion module.

Simulation of the stray field and MFM response

Several modules, differing by the algorithm used, simulate the stray field distribution in a plane above sample surface. None of them works with a general medium — e.g. using Poisson equation to solve the magnetic field distribution from the samle volume magnetisation — as this seems to be too computationally demanding for a general package like Gwyddion (at least for now). The modules are set up to simulate a few typical cases described in literature, where there are enough assumptions about sample magnetisation to make the stray field calculable by simple methods.

The module for handling *perpendicular media* is based on works of H. Hug [Hug98] and S. Vock [Vock11][Vock14]. The field distribution is calculated on the basis of known magnetisation distribution in a layer; in the z direction the magnetisation is constant for each position in the layer and changes only when we go in lateral directions. Initially, magnetisation is described as binary image (up/down magnetisation, defined by mask) and domain walls of given thickness can be added to it. The calculation is then done in Fourier space. Apart of the magnetic field distribution also results of using simple analytically known probe transfer functions can be provided (point charge, bar), and numerical derivatives of the resulting forces can be computed.

Assuming the surface charge distribution $+\sigma(x,y)$ at the top surface of the medium and opposite charge $-\sigma(x,y)$ at the bottom surface we can use Fourier analysis to get the field distribution. First we convert the surface charges to the Fourier space:

$$A_M(\mathbf{k}) = \int_{-\infty}^{\infty} \sigma(\mathbf{r}) \mathrm{e}^{\mathrm{i}\mathbf{k}\mathbf{r}} \mathrm{d}\mathbf{r}$$

Optionally we can even assume that in between of the domains there are Bloch walls of some width δ_w, convolving $\sigma(x,y)$ prior to the calculation with blurring operator

$$f(x,y) = 1 \Big/ \cosh \frac{\pi \sqrt{x^2+y^2}}{\delta_\mathrm{w}},$$

where the wall width can be expressed via exchange stiffness constant A and uniaxial anisotropy constant K_u as

$$\delta_\mathrm{w} = \pi\sqrt{A/K_\mathrm{u}}$$

Then we can calculate the Fourier components of the stray field in height z, generated from top surface and bottom surface of film of thickness d as

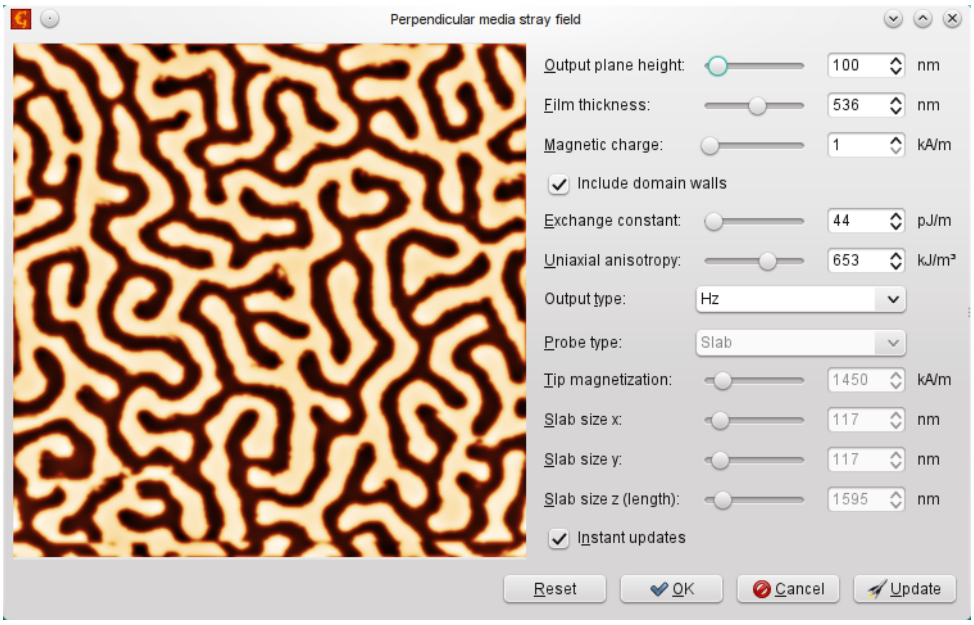
$$A^{z,d}_{H_{x,y}}(\mathbf{k}) = \mathrm{i}\frac{\mathrm{e}^{-kz}\left(1-\mathrm{e}^{-kd}\right)k_{x,y}}{2k}A_M(\mathbf{k})$$
$$A^{z,d}_{H_z}(\mathbf{k}) = \frac{\mathrm{e}^{-kz}\left(1-\mathrm{e}^{-kd}\right)}{2}A_M(\mathbf{k})$$

Via inverse Fourier transform we can then get the magnetic stray field above the sample:

$$H_{x,y}(\mathbf{r},z) = \frac{1}{4\pi^2}\int_{-\infty}^{\infty} A^{z,d}_{H_{x,y}}\mathrm{e}^{-\mathrm{i}\mathbf{kr}}\mathrm{d}\mathbf{k}$$
$$H_z(\mathbf{r},z) = \frac{1}{4\pi^2}\int_{-\infty}^{\infty} A^{z,d}_{H_z}\mathrm{e}^{-\mathrm{i}\mathbf{kr}}\mathrm{d}\mathbf{k}$$

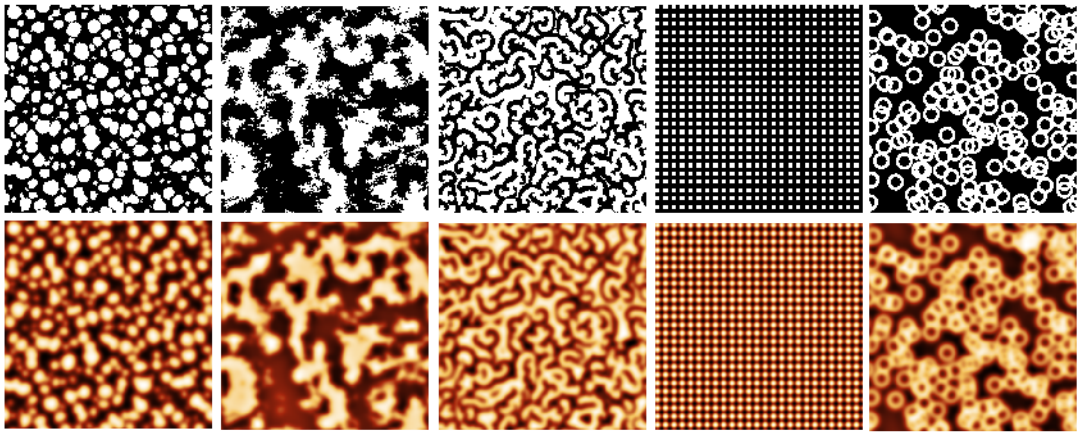
While still staying in the Fourier space, we can also calculate the force acting on a simple bar probe of cross-section $b_x \times b_y$, length L and magnetisation M_t, by multiplying the stray field components length $A^{z,d}_{H_z}$ via the analytical force transfer function:

$$\mathrm{FTF}(\mathbf{k}) = -\frac{4\mu_0 M_t}{k_x k_y}\left(1-\mathrm{e}^{-kL}\right)\sin\frac{k_x b_x}{2}\sin\frac{k_y b_y}{2}$$



Screenshot of the perpendicular media simulation module.

Since Gwyddion is not just able to handle measured data, but also to synthetise many different artificial surfaces, there are many ways how to play with the perpendicular media module, creating various fields on more or less realistic samples. The following figure shows few examples of such simulated masks and resulting data.



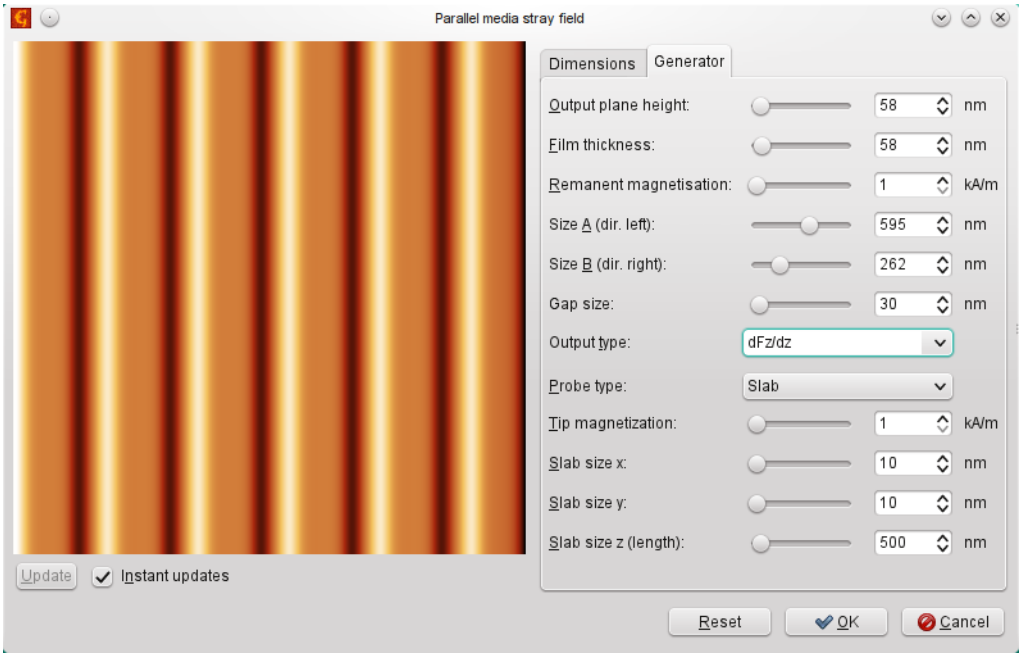
Result of data synthetic modules used for MFM perpendicular media simulation: columnar film, fractal roughness, Ising model domains, grating, doughnuts.

For *parallell media* the range of potential calculations is much smaller — based on an analytical model the field above left and right direction oriented stripes (similar to a hard-disc) can be computed. This is based on [Rugar90], giving the following equations for individual transitions

$$B_x(x,z) = \frac{\mu_0 M_r}{\pi}\left(\operatorname{atan}\frac{x(d+z)}{x^2+a^2+a(d+z)} - \operatorname{atan}\frac{xz}{x^2+a^2+az}\right)$$
$$B_z(x,z) = \frac{\mu_0 M_r}{2\pi}\ln\frac{x^2+(d+z+a)^2}{x^2+(z+a)^2}$$

where M_r, is the remanent magnetization of the magnetic layer, z the tip-sample separation, a is the transition area width, and d the layer thickness.

The stripes do not have to be of equal size. As a result, field intensities in x and z axes can be plotted, or the force in z based on known probe transfer functions (the same as for the perpendicular media) can be computed. Its derivatives can be computed as well (numerically).



Screenshot of the parallel media simulation module.

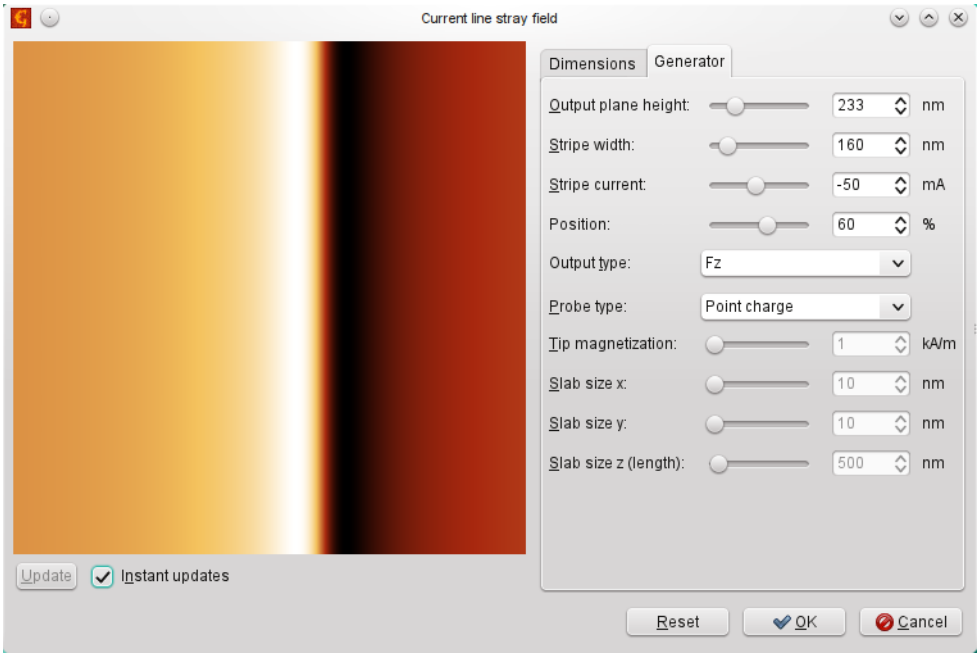
Similarily, analytical expressions can be used to simulate the current above a thin line with a electrical current passing through

it. The *current line* module can be used for this, which is based on the equation given by [Saida03]:

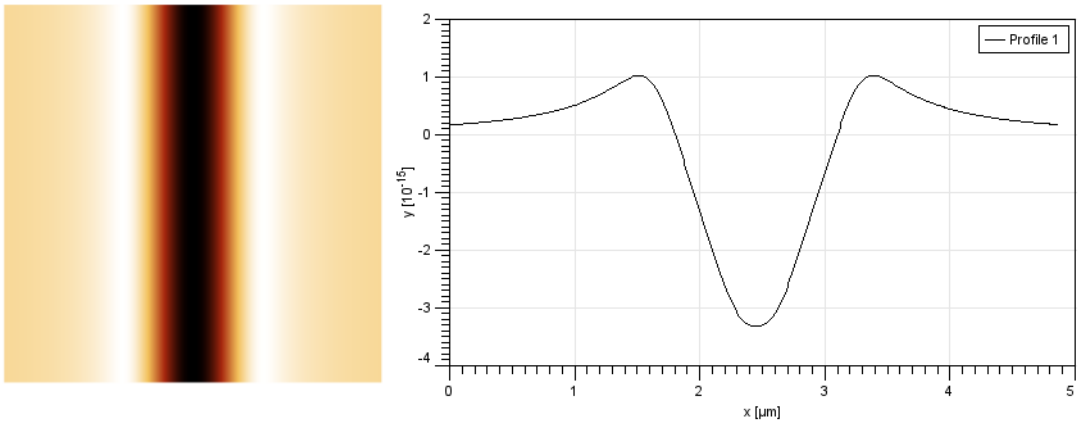
$$B_x(x,z) = \frac{\mu_0 I}{2\pi w} \operatorname{atan} \frac{wz}{z^2+x^2-w^2/4}$$
$$B_z(x,z) = \frac{\mu_0 I}{4\pi w} \ln \frac{(x-w/2)^2+z^2}{(x+w/2)^2+z^2}$$

for a stripe of width w in which the current I is flowing.

As all the simulation modules can be used also to add the result to existing data, by calling module multiple times we can also simulate a planar coil (or, more precisely, two lines with opposite directions of the current). This is demonstrated in the following figures.



Screenshot of the current line simulation module.



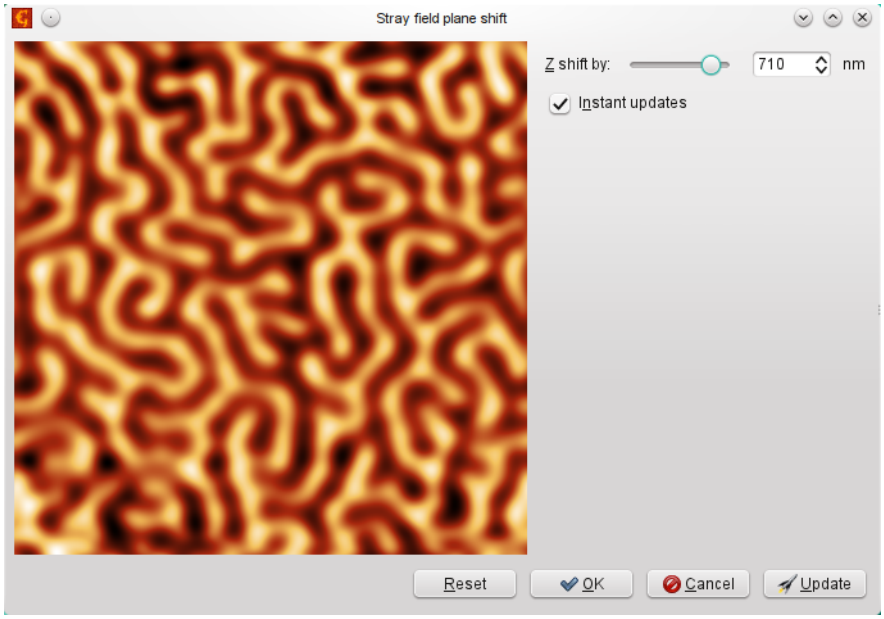
Stray field distribution above two parallel current lines with opposite current direction.

The algorithms for stray field calculation via Fourier transform can be also used to shift the field from one layer height to another. This works if we move further from the sample (blurring the data), but it does not work much when getting closer to the sample (the calculation diverges). Based on [Hug98] the procedure is as follows:

1. The field $H_z(\mathbf{r})$ is processed by Fourier transform to get its spectral representation $A_{H_z}(\mathbf{k})$,

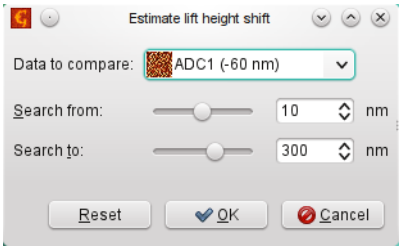
2. Spectral components are multiplied by the factor $\exp(kd)$ where d is the distance between the two planes (positive or negative),
3. Inverse Fourier transform is used to get back the field values in real space.

One can play with this algorithm in the *field shift* module as illustrated in the following figure.



Screenshot of the field shift module.

The same approach can be used for reverse operation — estmating the height difference between two data sets. Here a simple brute force search technique is used to find the shift in z direction that leads to best match between the two MFM responses. The user interface is shown in Fig. 8. Potential application is in searching for the real lift height difference instead of the reported one, however there are still potential caveats in real data if there are parasitic interactions.



Screenshot of the field shift estimation module.

Probe resolution and transfer function

Several ways have been proposed how to handle the issue of resolution of MFM probes. It is possible to analyse profiles across the perpendicular sample or utilise the power spectrum for probe resolution, e.g. by attempting to find where the power spectrum vanishes to noise level. See Transfer Function Guess for the transfer function estimation method currently implemented in Gwyddion. This technique is not, in fact, specific to MFM and can be used universally for transfer function TF (or point spread function, PSF) estimation in any SPM regime, provided that the ideal sample response can be recorded or simulated. The mathematical and algorithmic details on how the magnetic data are treated in the transfer function modules can be found in Ref. Nečas19.

Quantitative MFM toolchain example

To illustrate the whole procedure of obtaining quantitative MFM results we provide here an example of the transform function estimation on a known sample and data processing of data obtained on an unknown sample, based on this transfer function.

The starting point is to use the data measured on a known sample. Here a multilayer sample manufactured by IFW Dresden is used, for which the stray field above sample can be computed on basis of the known domains distribution and known sample

properties. Together with this, the "unknown" test sample data, provided by National Physical Laboratory are in the same input file available for download.

To go step by step and get the same results as provided in the processed file, the following procedure has to be done:

1. Convert calibration sample MFM phase data to MFM force gradient using Recalculate to Force Gradient with k 3.3 N/m and Q factor 226.
2. Add mask at 50% of the value using thresholding.
3. Calculate the effective magnetic charge using Perpendicular Media Stray field with 130 nm film thickness, 500 kA/m magnetic charge, 12 pJ/m exchange constant, 400 kJ/m^3 uniaxial anisotropy, 0 deg cantilever angle and output M_{eff}.
4. Calculate the TTF using Transfer Function Guess with Wiener filter method, 65×65 TTF size and Welch window.
5. Convert MFM phase data of the test sample to MFM force gradient using Recalculate to Force Gradient with k 3.3 N/m and Q factor 226.
6. Scale the TTF to the pixel size of the test sample.
7. Use Dimensions and Units to match exactly the pixel size of the test sample measurements.
8. Deconvolve the cropped TTF from MFM force gradient on test sample using Deconvolve, using the maxium of L-curve curvature for obtaining best sigma.
9. Divide the resulting effective magnetic charge by 2 to get the stray field H_z using Arithmetic.

References

[Hug98] H. J. Hug, B. Stiefel, P. J. A. van Schendel, A. Moser, R. Hofer, S. Martin, H.-J. Gutherodt, S. Porthun, L. Abelmann, J. C. Lodder, G. Bochi and R. C. O'Handley: J. Appl. Phys. (1999) Vol. 83, No. 11

[Vock11] S. Vock, Z Sasvari, C. Bran, F. Rhein, U. Wolff, N. S. Kiselev, A. N. Bogdanov, L. Schiltz, O. Hellwig and V. Neu: IEEE Trans. on Magnetics, 47 (2011) 2352

[Vock14] S. Vock, C. hengst, M. Wolf, K. Tschulik, M. Uhlemann, Z. Sasvari, D. Makarov, O. G. Schmidt, L Schultz and V. Neu: Appl. Phys. Lett 105 (2014) 172409

[Rugar90] D. Rugar, H. J. Mamin, P. Guethner, S. E. Lambert, J. E. Stern et al.: J. Appl. Phys. 68 (1990) 1169

[Saida03] D. Saida, T. Takahashi: Jpn. J. Appl. Phys. 42 (2003) Pt. 1, No. 7B

[Cambel11] V. Cambel, D. Gregusova, P. Elias, J. Fedor, I. Kostic, J. Manka, P. Ballo: Journal of Electrical Engineering, 62 (2011) 37–43

[Necas19] D. Nečas, P. Klapetek, V. Neu, M. Havlíček, R. Puttock, O. Kazakova, X. Hu, L. Zajíčková, Scientific Report, 9 (2019) 3880

4.22 Scanning Microwave Microscopy

Scanning Microwave Microscopy is based on measurement of reflection coefficients of microwaves applied to a conductive tip. Using vector network analyzer or similar hardware we want to obtain the reflection coefficient S_{11} and, out of it, to determine tip-sample impedance, capacitance, or any other quantity related to it.

Here, we describe two modules implementing modified short-open-load calibration procedure described in Ref. [1]. It is based on measurement of a sample with areas of known capacitances, e.g. small capacitors acting as reference samples[2], and by comparison of theoretical and measurement reflection coefficients it establishes complex calibration coefficients that can be used for SMM calibration, i.e. to convert measurement on unknown samples to quantitative results.

SMM calibration

Goal of this module is to use data measured on a known sample to determine three complex calibration coefficients e_{00}, e_{01} and e_{11} containing infomration about directivity, reflection tracking, source match, merged with information about the local electrical conditions around the tip apex[3]. The coefficients relate the measured reflection coefficients S_{11m} to intrinstic reflection coefficients S_{11} as

$$S_{11} = \frac{S_{11m} - e_{00}}{e_{01} + e_{11}(S_{11m} - e_{00})}$$

The intrinstic reflection coefficients are related to complex tip-sample impedance Z_{tip} and reference impedance Z_{ref} as

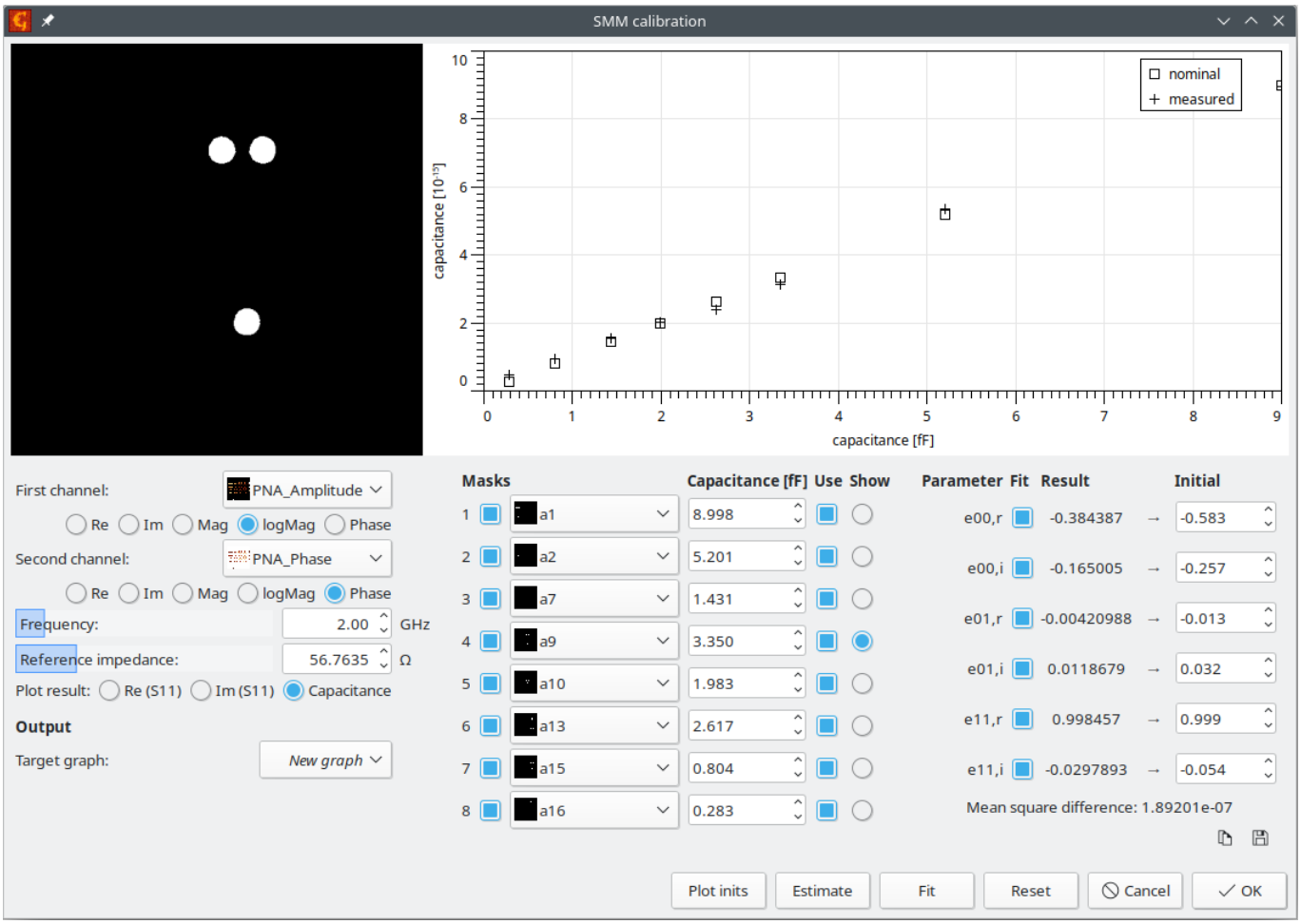
$$Z_{tip} = Z_{ref} \frac{1+S_{11}}{1-S_{11}}$$

For a capacitor reference sample, the tip-sample impedance can be calculated from its known capacitance (obtained from sample manufacturer or via numerical modeling) and angular frequency of the microwaves as

$$Z_{tip} = \frac{1}{i\omega C}$$

The module uses the above three equations to determine the calibration coefficients, either by direct solution of the equations for three complex measurements (when only three sets are provided) or by fitting all the available data (when more sets are provided). Direct solution is also used for an estimate, using the minimum, maximum and middle of the provided capacitances.

Apart of two images representing complex data from VNA (either a pair of real + imaginary or module + phase images), user has to provide masks covering the individual surface areas where the capacitance is known and values of this capacitance. These masks have to be set up before running the module, either using thresholding or some of the mask edit tools, followed by mask extraction, so it resides in the Data Browser as an individual channel. User can check which of the provided sets (mask + capacitance) will be used for calculation or fitting of the calibration coefficients.

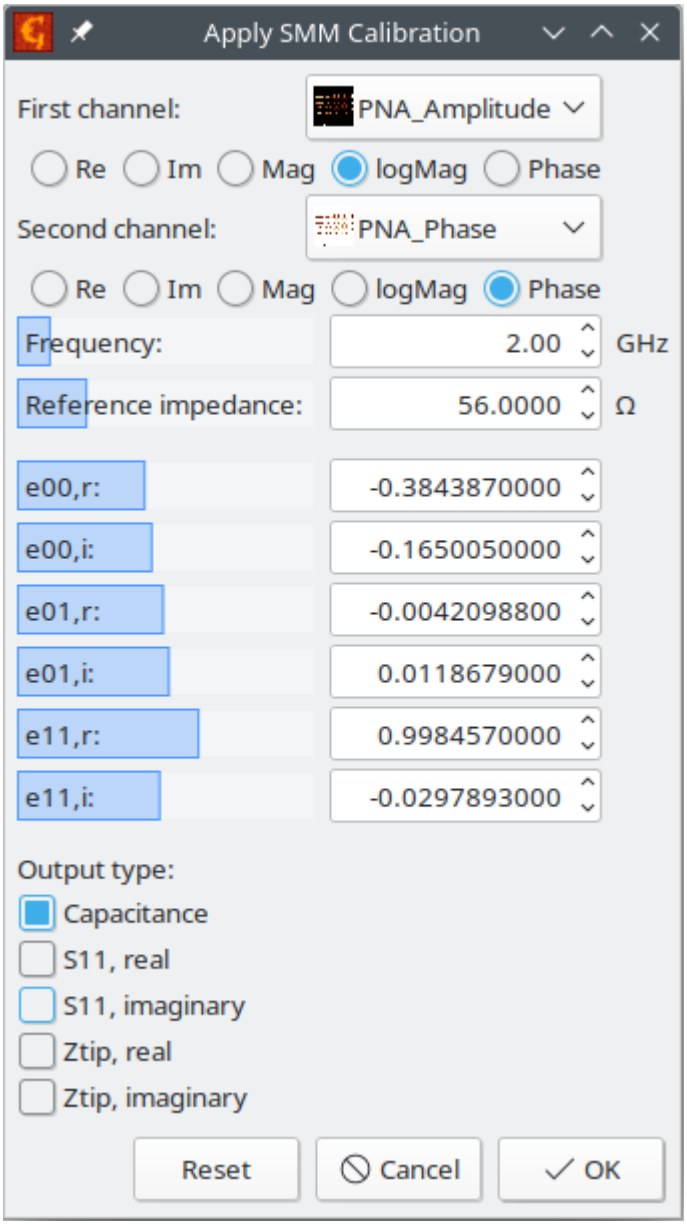


SMM calibration module graphical user interface

To test the module, use a sample file with pre-processed data, including with few extracted masks, corresponding to capacitances in the above figure. This sample file is based on measurements performed by Damien Richert, Laboratoire national de métrologie et d'essais and shows MC2 capacitance sample.

Apply SMM Coefficients

This module applies known complex calibration coefficients to the measured data, using the same methodology and equations as discussed in the preceding section. As an input, complex data from VNA (either a pair of real + imaginary or module + phase images) are used. When data should be converted to capacitance, the microwave frequency and reference impedance has to be supplied.



SMM calibration application module graphical user interface

References

[1] Hoffmann, J.; Wollensack, M.; Zeier, M.; Niegemann, J.; Huber, H. A Calibration Algorithm for Nearfield Scanning Microwave Microscopes. In Proceedings of the 12th IEEE International Conference on Nanotechnology, IEEE-NANO, Birmingham, UK, 20–23 August 2012; pp. 1–4. doi:10.1109/NANO.2012.6322116

[2] Piquemal, F.; Morán-Meza, J.; Delvallée, A.; Richert, D.; Kaja, K., Progress in Traceable Nanoscale Capacitance Measurements Using Scanning Microwave Microscopy. Nanomaterials 2021, 11, 820. doi:10.3390/nano11030820

[3] A. Buchter, J. Hoffmann, A. Delvallée, E. Brinciotti, D. Hapiuk, Ch. Licitra, K. Louarn, A. Arnoult, G. Almuneau, F. Piquemal, M. Zeier, F. Kienberger, Review of Scientific Instruments 89, 023704 (2018). doi:10.1063/1.5015966

4.23 Graph Processing

Many of the Gwyddion data processing modules produce graph as an output. Graphs can be exported into text files or further analysed within Gwyddion by several graph processing modules. These modules can be found in the Graph menu in the Gwyddion main window. Note that the number of graph modules is quite limited now and consists of basic modules for doing things that are very frequent within SPM data analysis. For more analytical tools you can use your favourite graph processing program.

In this section the graph modules present in Gwyddion are briefly presented.

Basic Operations 

First of all zooming and data reading functions are available directly in the graph window:

- Logarithmic axes – horizontal and vertical axes can be switched between linear and logarithmic using the logscale buttons. Switching to logarithmic scale is possible only for positive values (either on abscissa or ordinate).
- Zoom in and zoom out – after selecting zoom in simply draw the area that should be zoomed by mouse. Zoom out restores the state where all data can be seen.
- Measure distances – enables user to select several points within the graph and displays their distances and angles between them.

Graph Flip

Graph Flip flips the graph vertically. In other words the ordinate is inverted, whereas the abscissa is kept untouched. The functions directly modifies the curve data; it does not just change the presentation.

Graph Invert

Graph Invert flips the graph horizontally. In other words the abscissa is inverted, whereas the ordinate is kept untouched. The functions directly modifies the curve data; it does not just change the presentation.

Graph Cut 

Graph Cut is a very simple module that cuts graph curves to the selected range (either given numerically or on the graph with mouse) and creates a new graph. If *Cut all curves* all curves are cut and put to the new graph; otherwise just the one chosen curve.

Graph Level 

Graph level is also a simple module that currently performs linear fit of each graph curve and subtracts the fitted linear functions from them.

Graph Align 

Graph align shifts the graphs curves horizontally to maximise their mutual correlations, i.e. match common features in the curves. This is useful in particular for comparison of profiles taken in different locations.

Graph Merge and Average

Two simple functions combine data of all curves to one, which is added to the plot. They differ slightly by data averaging approach.

Merge Curves simply collects the data from all curves. It only merges two data points if the abscssae are the same, averaging the ordinates. The resulting curve thus usually oscillates between points belonging to the input curves if they are slightly vertically shifted.

Average Curves attempts to combine close points, averaging both ordinates and abscissae. Usually the resulting curve has a similar (or somewhat larger) number of points to one input curve, not all of them together. It should be also much smoother than for the simple merge, although sometimes there can be still visible oscillations.

Logscale Transform

The graph window controls allow switching between linear and logarithmic axes. However, for the fitting of power-law-like dependencies it is often useful to physically transform the data by taking logarithms of the values. The logscale transform graph function performs such transformation. You can choose which axis to transform (x, y or both), what to do with non-positive values if they occur and the logarithm base. A new graph is then created, with all curves transformed as requested.

Exporting Graph Curves 

Graph curve data can be exported to text files using *Export Text*. The export dialogue allows choosing several style variations that particular other software may find easier to read. Options *Export labels*, *Export units* and *Export metadata* allow adding informational lines before the actual data. This can be very useful for reminding what the file contains, but may cause problems when the file is read in other software.

Option *POSIX number format* enforces standard machine-readable scientific number format with decimal dot. Otherwise the values are written according to locale settings (office-style software may like that; scientific software generally does not).

The other important option that influences the structure of the entire file is *Single merged abscissa*. By default individual curves are written to the file sequentially, separated by empty lines. When this checkbox is enabled the curve export writes one multi-column table with data of all curves and a single abscissa in the first column. If the curves are not sampled identically, some rows will of course contain values only for some curves. Exported file with two separated curves can look like

```
0.1 3.32e6
0.2 3.80e6
0.4 4.15e6

0.0 11.1
0.3 9.66
0.4 9.70
```

while with single merged abscissa the same data would be saved

```
0.0 --- 11.1
0.1 3.32e6 ---
0.2 3.80e6 ---
0.3 --- 9.66
0.4 4.15e6 9.70
```

It is also possible to export a vector (EPS) or bitmap (PNG) rendering of the graph using *Export PostScript* or *Export Bitmap*. However, these options are rather rudimentary. Gwyddion is not a dedicated plotting software and if you want nice graphs use one instead – for instance gnuplot or matplotlib.

Statistics 

Graph statistics display summary information about entire graph curves or selected ranges. The dialogue shows two main groups of quantities that are calculated differently.

Simple Parameters are calculated from the set of ordinate values, without any regard to abscissae. This is important to keep in mind when the curve is sampled non-uniformly, i.e. the intervals between abscissae differ, possibly a lot. A part of the curve which is sampled more densely contains relatively more points and thus also influences the result more. The available parameters include elementary characteristics with the same meaning as for two-dimensional data. Several of them also coincide with basic roughness parameters calculated by the Roughness tool.

On the other hand, *Integrals* are obtained by integration, utilising the trapezoid rule (or a similar approximation). Therefore, longer intervals contribute more to the results. Available quantities include:

Projected length The length of the selected interval (or the entire abscissa range if no interval is selected).

Developed length Sum of lengths of linear segments connecting curve points.

Variation Integral of absolute value of the derivative – calculated as the sum of absolute values of ordinate differences.

Average value Area under the curve divided by the projected length.

Area under curve The total integral (sum of positive and negative area).

Positive area Integral of portions of the curve where it is positive.

Negative area Integral of portions of the curve where it is negative.

Root mean square Integral of squared values divided by the projected length.

Statistical Functions

One-dimensional statistical functions are calculated for graphs using the same definitions as for images. They are described in the Statistical Functions tool documentation. The available functions include height and angle distributions, autocorrelation function, height-height correlation function and spectral density. They can be calculated for the selected curve or for all curves at once if *All curves* is enabled.

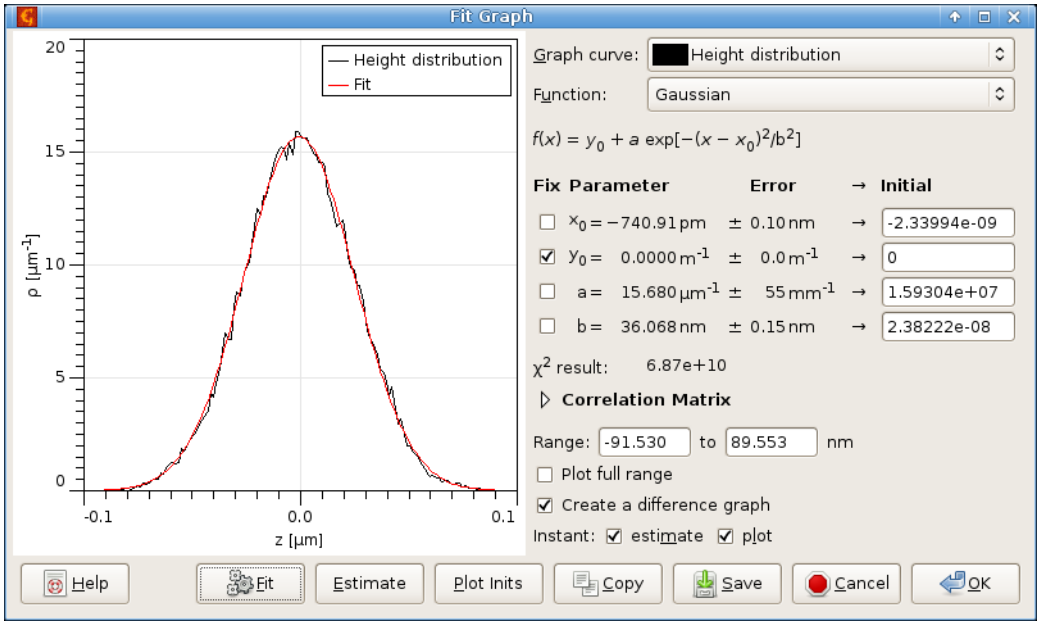
A major difference between images and graphs is that graph curves do not need to have uniformly spaced abscissa values (uniform sampling). In such case the curve is resampled within the calculation of the statistical functions to uniform sampling. The default sampling step is such that the number of points is preserved. However, it can be modified by *Oversampling* which gives how many more points the resampled curve should have compared to the graph curve. Occasionally it may be useful to use oversampling larger than 1 even for regularly sampled graph curves.

Function Fitting 

The curve fitting is designed namely for fitting of statistical functions used in roughness parameters evaluation. Therefore most of the available functions are currently various statistical functions of surfaces with Gaussian or exponential autocorrelation functions. Nevertheless it also offers a handful of common general-purpose functions. See the list of fitting functions.

Within the fitting module you can select the area that should be fitted (with mouse or numerically), try some initial parameters, or let the module to guess them, and then fit the data using Marquardt-Levenberg algorithm.

As the result you obtain the fitted curve and the set of its parameters. The fit report can be saved into a file using *Save* button. Pressing *OK* button adds the fitted curve to the graph, if this is not desirable, quit the dialogue with *Cancel*.



Curve fitting module dialogue

Force-Distance Curve Fitting 

The module for fitting of force-distance curves is very similar to the general curve fitting module, it is just specialized for force-distance curves. Currently, the module serves for fitting jump-in part of force-distance curve (representing attractive forces) using different models:

- van der Waals force between semisphere and half-space
- van der Waals force between pyramid and half-space

- van der Waals force between truncated pyramid and half-space
- van der Waals force between sphere and half-space
- van der Waals force between two spheres
- van der Waals force between cone and half-space
- van der Waals force between cylinder and half-space
- van der Waals force between paraboloid and half-space

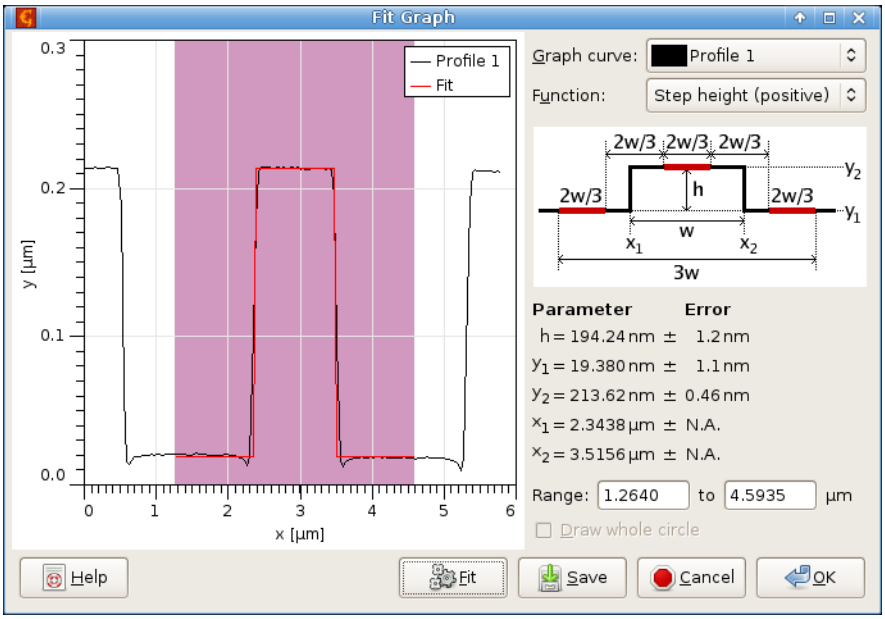
Note that the curve being fitted must be a real force-distance curve, not a displacement-distance or sensor-distance curve. Recalculation of cantilever deflection into force should be done before calling this module.

Also note that for small cantilever spring constants the amount of usable data in attractive region is limited by effect of jumping into contact.

Critical Dimension

Critical dimension module can be used to fit some "typical" objects that are often found while analysing profiles extracted from microchips and related surfaces. These objects are located in the graph and their properties are evaluated.

The user interface of this module is practically the same as of the graph fit module.



Critical dimension module dialogue.

DOS Spectrum

DOS spectrum module intended to obtain Density-of-States spectra from I-V STM spectroscopy. It calculates

$$\left| \frac{\frac{\mathrm{d}I}{\mathrm{d}U}}{\frac{I}{U}} \right| (U)$$

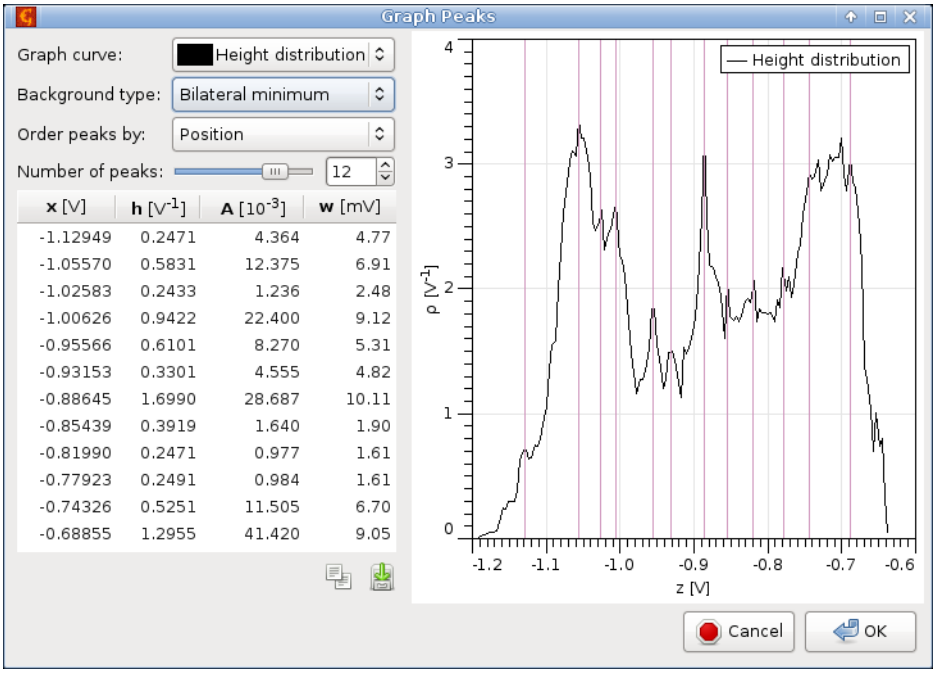
and plots it as graph.

Find Peaks

The most prominent peaks in graph curves can be automatically located with subsample precision. You can specify the number of most prominent peaks as *Number of peaks* and the function will mark them on the graph. Peak prominence depends on its height, area and distance from other peaks. Usually the function's idea what are the most prominent peaks agrees quite well with human assessment. If you disagree with the choice you can ask for more peaks and ignore those you do not like. It is also possible for locate negative peaks, i.e. valleys, by enabling option *Invert (find valleys)*.

A table of all the peaks is displayed in the left part, sorted according to *Order peaks by*. Ordering by position means peaks are listed as they are displayed on the graph from left to right. Prominence order means more significant peaks are listed first.

Several basic characteristics are displayed for each peak: position (abscissa) x, height h, area A, and width (or variance) w. The position is calculated by quadratic subpixel refinement of the peak maximum. The remaining quantities depend on how the peak background is defined. Possible choices include *Zero* meaning peaks base is always considered to be zero, and *Bilateral minimum* meaning the peak background is a step function passing through the nearest curve minima on the left and right side of the peak.



Screenshot of automated graph peak location for a height distribution plot.

Period/pitch

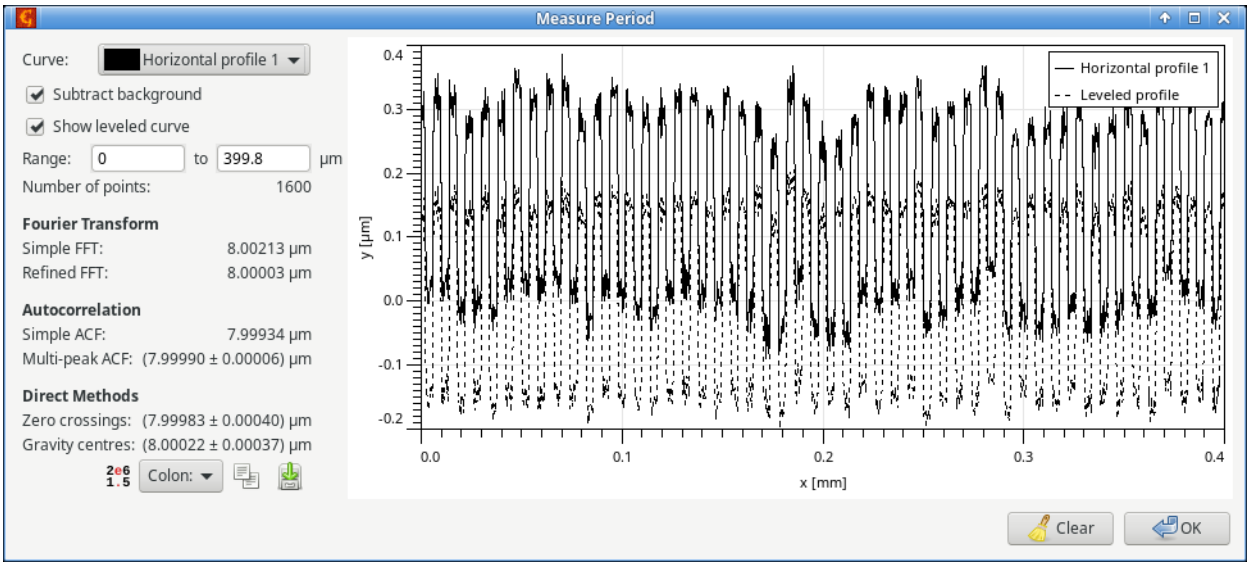
Many methods for measurement of the period/pitch of periodic profiles such as gratings have been proposed. The graph period measurment module implements several of them (see reference [1] for their description). For reasonably well measured gratings all the good methods

- Refined Fourier transform
- Multi-peak ACF
- Zero crossings
- Gravity centres

should give comparable results, although they differ in sensitivity to various artefacts in the data. Refined Fourier transform is the most robust for odd shapes of the repeating unit. However, it does not provide any error estimate.

Two elementary methods are also included, simple Fourier transform and simple ACF, which just find the position of the main peak in the power spectrum or ACF. They should not be used for evaluation and are present mainly for completness.

If the profile has been already correctly levelled and the zero line selected, the function can evaluate the profile without any preprocessing. Otherwise, *Subtract background* should be checked to enabled background removal based on a robust procedure (see again [1]). In such case *Show levelled curve* can be used to display the preprocessed profile alongside with the unmodified data.

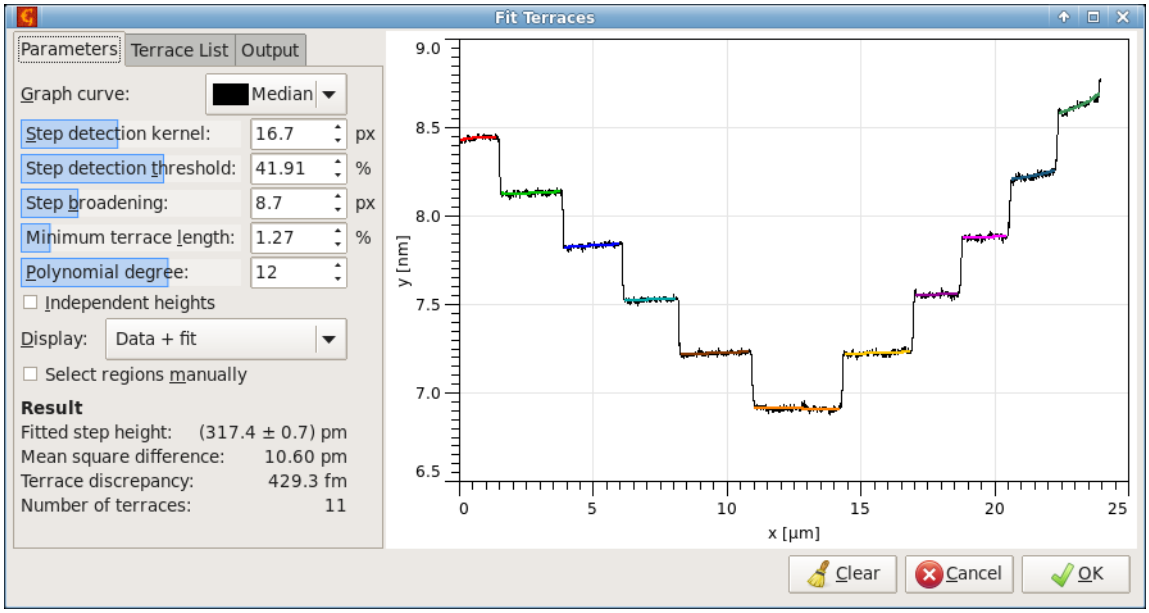


Measurement of grating pitch.

Terraces 

Terrace or amphitheatre-like structures can be measured using one-dimensional profiles, similarly as they are measured from image data in Terraces. The graph and image modules are almost identical. Therefore, the following only highlights the main differences:

- Step detection kernel and broadening are one-dimensional. They are still measured in pixels – which correspond to average spacing between curve points. Minimum terrace area is replaced by minimum length, measured as the fraction of the total abscissa range.
- There is no masking option. Instead, terraces can be marked manually on the graph using mouse when *Select regions manually* is enabled.
- There is an additional choice in *Display* menu, *Step detection*. It shows the results of the edge detection filters and marks the selected threshold using a dashed red line. This is one of the most useful graphs for parameter tuning.
- Pixel area A_{px} is replaced with N_{px}, the number of curve points the terrace consists of.



Terrace measurement from a profile, showing marked terraces for an amphitheatre structure formed by atomic steps on silicon.

References

[1] D. Nečas, A. Yacoot, M. Valtr, P. Klapetek: Demystifying data evaluation in the measurement of periodic structures. Measurement Science and Technology 34 (2023) 055015, 10.1088/1361-6501/acbab3

4.24 Volume Data Processing

Gwyddion currently provides several basic functions for visualisation of volume data and extraction of lower-dimensional data (images, curves) from them. A few specialised functions, focused on processing volume data as curves (or spectra) in each pixel, are also available. They are all present in the *Volume Data* menu of the toolbox.

Volume data are often interpreted in Gwyddion as a set of curves, each attached to one pixel in the xy plane, or alternately stack of images along the z axis. This means the volume data functions may treat the z axis specially. If you intend to import volume data with two spatial and one special dimensions to Gwyddion make sure that the special axis corresponds to z.

Basic Operations

Elementary operations with volume data include:

Dimensions and Units  It changes the physical dimensions, units or value scales and also lateral offsets. This is useful to correct raw data that have been imported with wrong physical scales or as a simple manual recalibration of dimensions and values.

Invert Value  This function inverts the signs of all values in the volume data.

Extract Preview The preview image of the volume data is copied to a new image in the file. Frequently the preview is an image that could be equivalently obtained using Cut and Slice or Summarize Profiles. And using these functions create images of well-defined quantities. However, if you simply want to get the preview image, whatever it happens to be, this function will do just that.

Swap Axes  Since the z axis is treated somewhat differently than the xy plane, it can be useful to change the roles of the axes. This function rotates and/or mirrors the volume data so that any of the current x, y and z axes will become any chosen Cartesian axis in the transformed data.

The dialogue ensures the transformation is non-degenerate. When you select a transformation that would be degenerate, another axis is adjusted to prevent this. Otherwise, you can perform any combination of mirroring and rotations by multiples of 90 degrees around Cartesian axes.

If the volume data have a z-axis calibration and z becomes some other axis the calibration will be lost. A warning is shown when this occurs.

Z Calibration  When the z-axis is non-spatial the sampling along it may be sometimes irregular, unlike the imaging axes x and y that are always regular. In Gwyddion this is represented by associating certain one-dimensional data with the z-axis, called the Z calibration.

This function can display the Z calibration, remove it, extract it as a graph curve, take calibration from another volume data or attach a new calibration from a text file. For attaching a calibration, each line of the file must contain one value specifying the true z-axis value for the corresponding plane and, of course, the number of lines must correspond to the number of image planes in the volume data.

Image and Profile Extraction 

Profiles along any axis and image slices in planes orthogonal to any axis can be extracted with *Volume Data* → *Cut and Slice*.

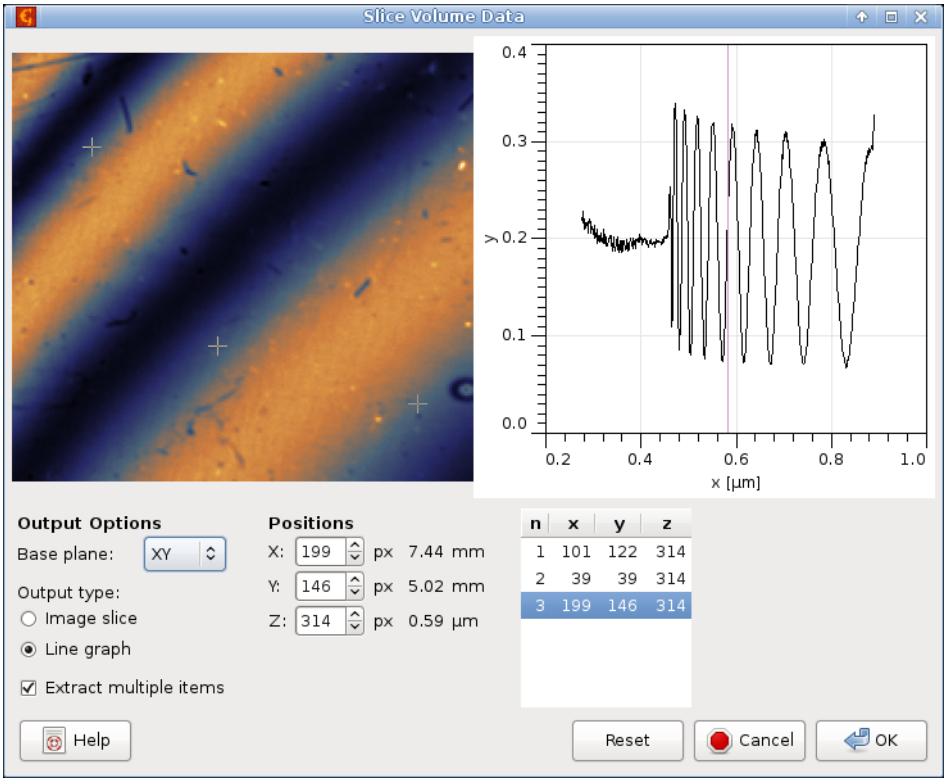
The image slice in the selected plane, called *Base plane*, is always shown in the left part and the profile along the remaining orthogonal axis is shown in the right part of the dialogue. You can change the where the line profile is taken by moving the selected point in the image. Conversely, the image plane location can be moved by selecting a point in the profile graph. Both can be also entered numerically as X, Y and Z (in pixel coordinates).

You can choose between image and profile extraction freely by changing *Output type*. Changing the output type only influences whether the module creates the image on the left or the profile graph on the right when you press *OK*.

It is also possible to take multiple profiles or image slices at the same time. If *Extract multiple items* is enabled a list of selected points will appear on the right. In this case the output type influences the selection. In the *Image slice* mode you can select only

one point in the image (determining the profile shown in the graph) but multiple points in the graph, each determining one output image plane. Conversely, in the *Line graph* mode you can select only one point in the graph (determining the image plane shown) but multiple points in the image, determining where the line profiles will be taken.

Switching betwen the output types when mutli-item extraction is enabled reduces the selected coordinate set to a single point. You can also use the *Reset* button to reduce the selection to the default single point if you want to start afresh.



Screenshot of the Cut and Slice volume data processing module, showing the simultaneous extraction of three line profiles along the z-axis.

Text Export

Volume data can be exported to text files in several formats using *Volume Data* → *Export Text*. The possible output formats are

VTK structured grid Format indented for direct loading into VTK-based software such as ParaView. The values are dumped as a block of `STRUCTURED_POINTS`.

One Z-profile per line Each line of the output file consists of one z-profile. There are as many lines as there are points in each xy plane (image) of the volume data. The profiles are ordered by row, then by column (the usual order for image pixels).

One XY-layer per line Each line of the output file consists of one xy image plane, stored by row, then by column (the usual order for image pixels). There are as many lines as there are z-levels in the volume data. The layers are stored in order of increasing z.

Blank line separated matrices Each line of the output file consists of one image row in one xy plane. After one entire plane is stored a blank line separates the next plane. The layers again stored in order of increasing z.

Line Profile Characterisation

Basic overall characterisation of profiles along the z-axis can be performed using *Volume Data* → *Summarize Profiles*. This function creates an image formed by statistical characteristics of profiles taken along the z-axis. The set of available statistical quantities is the same as in the Row/column statistics tool.

The dual image/graph selection generally works very similarly to Image and profile extractions. The main difference is that an interval can be selected in the graph, defining the part of the image stack from which the statistical characteristics will be calculated. The interval can be also entered numerically using the *Range* entries.

The image on the left shows the calculated characteristics and corresponds to the output of the module. Selecting a different point in the image changes the profile displayed in the graph, which can be useful for choosing a suitable range. The value for the selected profile is displayed below the statistical quantity selector. However, choosing a profile does not influence the output in any way.

Plane Characterisation 

Basic overall characterisation of *xy* planes can be performed using *Volume Data* → *Summarize Planes*. This function creates a graph formed by statistical characteristics of individual *xy*-planes in the volume data or their parts. The set of available quantities is a subset of those calculated by Statistical quantities tool. A few quantities can take some time to calculate for all layers.

The dual image/graph selection generally works very similarly to Image and profile extractions. The main difference is that a rectangular region can be selected in the image plane, defining a rectangle from which the statistical characteristics will be calculated. The rectangle can be also entered numerically using the *Origin* and *Size* controls.

The graph on the right shows the calculated characteristics and corresponds to the output of the module. Selecting a different position in the graph changes the *xy* plane displayed in the image, which can be useful for choosing a suitable region. The value for the selected plane is displayed below the statistical quantity selector. However, choosing a plane does not influence the output in any way.

Arithmetic 

Arithmetic with volume data works exactly the same as image data arithmetic, including the same expression syntax.

The preview shows the average over all levels and the set of automatic variables is slightly different:

Variable	Description
`d1`, ..., `d8`	Data value at the voxel. The value is in base physical units, e.g. for current of 15 nA, the value of `d1` is 1.5e-8.
`x`	Horizontal coordinate of the voxel (in real units). It is the same in all volume data due to the compatibility requirement.
`y`	Vertical coordinate of the voxel (in real units). It is the same in all volume data due to the compatibility requirement.
`z`	Depth (level) coordinate of the voxel (in real units). It is the same in all volume data due to the compatibility requirement.
`zcal`	Calibrated *z*-coordinate of the voxel (in real units) if the volume data have *z*-axis calibration (see Z calibrations).

K-means and K-medians clustering 

Techniques of imaging spectroscopy like F-D in QNM (Quantitative Nanomechanical Mapping) applications, I-V in semiconductor engineering, Raman in material characterization require to measure spectrum per each point of data grid on sample surface. Working with obtained array is not so simple, as we need to work with thousands of spectra and analyze each of them. If the sample of interest has limited number of regions with very similar spectra inside each region, some technique grouping similar spectra together can be of great help. One possible way of doing so is using clustering analysis. In Gwyddion currently two methods is implemented: K-means and K-medians.

Both algorithms are intended to find *K* clusters with similar spectra inside the cluster and maximally different between clusters. So *Number of clusters* is how many clusters *K* you want to obtain in the result. *Convergence precision digits* and *Max. iterations* regulate convergence criteria of algorithms that stop either if required precision of cluster centers position is achieved or allowed number of convergence cycles is exceeded. Higher precision require more steps to achieve, second limit is mostly intended to get rid of endless loops if the precision is to high for this set of data.

Normalize is somewhat experimental technique that qives nice results for imaging spectroscopy. If you are not interested in spectral intensities and want to cluster data by similarity of middle-frequencies of spectral features (it is typical, for example, for Raman microscopy), than enable this checkbox. It removes low frequency background by substraction of minimum withing the moving window, and then normalizes spectrum to make average intensity be equal 1.0. Both modules outputs two datafields: one shows to which cluster belongs the current spectrum, another shows error — difference between the current spectrum and center of corresponding cluster. If normalization is enabled than third datafield shows values the spectral intensities was divided by during normalization (after low-freq. filtering). Also graph with spectra corresponding to the centers of clusters is extracted.

Algorithms beside this two modules are based on usual unsupervized learning clustering: K-means and K-medians, correspondingly. We consider each spectrum (graph extracted along z-axis) to be the point in multidimensial space with the number of dimensions equal number of data point in the graph. We define distance between points as square root from sum of squared differencies along each direction (L^2-norm). Than we initialize both algorithms by choosing random K points from the volume data as cluster centers. Than the usual two step algorithm is applied: we assign each point to the cluster whose center is nearest to the point and then move centers of the clusters. The difference between two modules is how to calculate the cluster center: it is mean value of cluster points for K-means and median along each direction for K-medians. The results from the last iteration are returned back to the program.

Remove outliers for K-means modifies algorithm for calculation of cluster centers to include only datapoints within *Outliers threshold* multiplied by data variation σ for each cluster. It removes single distinct bogus data points (like spikes, cosmic rays in Raman spectroscopy and so on) from calculation, making cluster centers somewhere more clean and distinct and also can shift border between intermixing clusters to more correct place between two centers of maximal point densities not disturbed by far outliers.

Magnetic force microscopy

A few volume data functions are dedicated to MFM data processing. They are generally analogous to their image counterparts.

Conversion of MFM data to force gradient is exactly the same for volume and image data. Therefore, the summary of image MFM data conversion applies to volume data unchanged.

The transfer function estimation *Volume Data* → *Transfer Function Guess* is also quite similar to image transfer function estimation. The TF is estimated for each level in the volume data and its various properties plotted as graphs. Which graphs are plotted can be selected in the *Output Options* tab. The regularisation parameter σ can be the same for all levels, or it can be estimated for each level by enabling *Estimate sigma for each level*. The computation then takes notably longer. For the *Least squares* method, even the transfer function size can be estimated for each level when *Estimate size for each level* is enabled. The size given in the dialogue then determines the size of the output – and thus maximum possible size. Smaller transfer functions are extended with zeros to this size.

The stray field consistency check *Volume Data* → *Stray Field Consistency* recalculates the field to a different height similarly to MFM field shift. However, since the volume data contains images corresponding to many different levels, it is possible to calculate the field for each and compare it to the calculated field. And this is what the module does.

4.25 Volume Data Processing: Image stacks

Volume data can be also used to represent stacks of images, i.e. coming from high-speed Scanning Probe Microscopy, representing multiple measurements of the sample, either on the same place or when moving laterally. In this case the data should be loaded as stack of images along the z axis. Such data can be processed by specialized functions available in the *Volume Data* menu of the toolbox, mostly described in this part of the User Guide. On top of it, the general functions for handling Volume data can be used as well.

XY Plane Correction

A few simple corrections or adjustments can be applied automatically to each xy plane in the volume data.

Volume Data → *Correct Data* → *XY Plane Level* applies plane levelling to each plane. In other words, it subtracts the mean plane from all images formed by xy planes. Each plane is treated separately.

Volume Data → *Correct Data* → *XY Mean Plane Level* applies plane levelling using the same plane, the mean one, to each plane in the image stack. It calculates a single mean plane from all the images formed by xy planes, and subtracts the mean plane from all the images.

Volume Data → *Correct Data* → *XY Facet Level* applies facet leveling to all the images formed by xy planes, see description of the Facet Level algorithm for more details. Each plane is treated separately.

Volume Data → *Correct Data* → *XY Flatten Base* applies flatten base algorithm to all the images formed by xy planes, see description of the Flatten Base algorithm fro more details. Each plane is treated separately.

Volume Data → *Correct Data* → *Step Line Correction* applies step line correction algorithm to all the images formed by xy planes, see description of the Step Line Correction module for images. Each plane is treated separately.

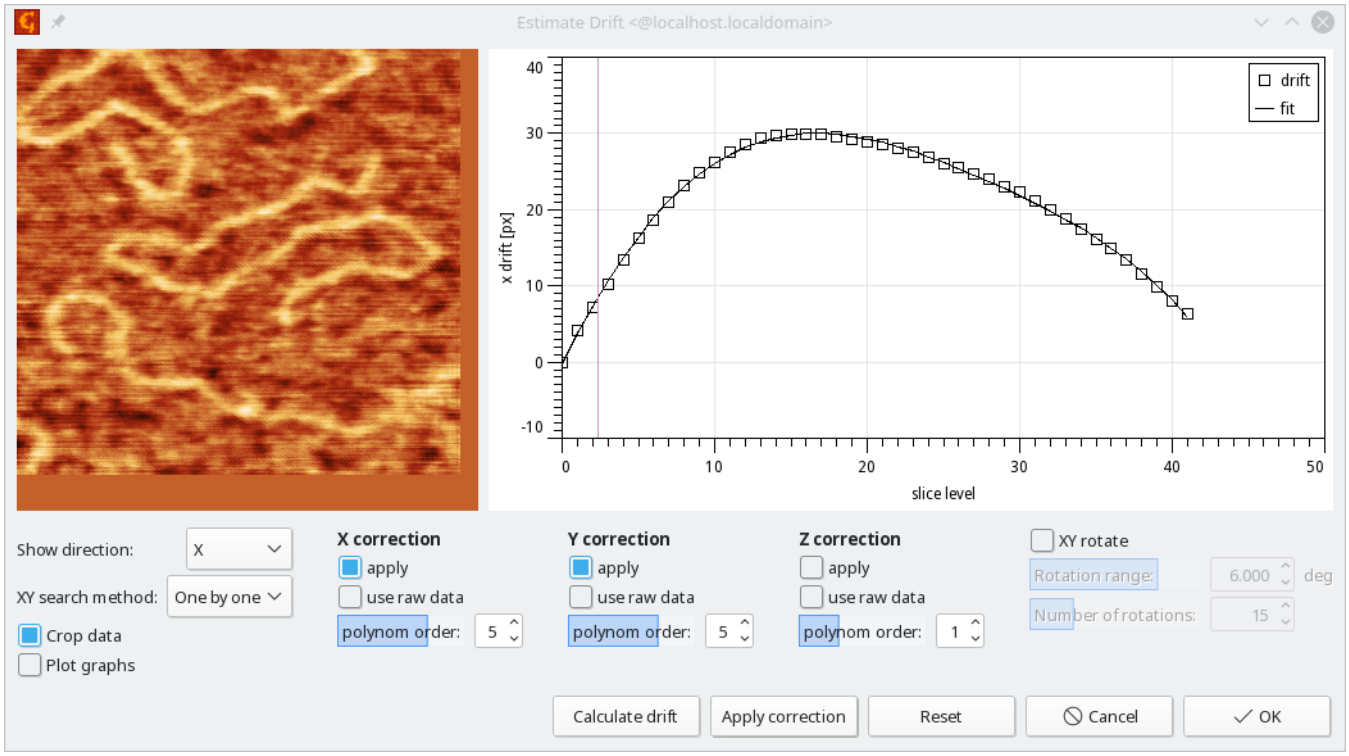
Volume Data → *Correct Data* → *XY Plane Outliers* replaces the outliers in each xy plane by the mean value of this plane. Outliers are defined here as values lying farther than three standard deviations from the mean.

Volume Data → *Correct Data* → *1D FFT Filtering* performs 1D FFT filtering in each xy plane. The power spectrum shown in the graph can be calculated for every plane individually, or it can be averaged from all the planes. By changing the preview level,

one can go through all the planes and cross-check the filtering impact. Mouse selection in the graph and the suppression methods have same logic as in the 1D FFT Filtering module for images.

Volume Data → *Correct Data* → *Scars* performs detection and removal of scars (fast scanning axis feedback loop errors) in each *xy* plane. Each plane is treated separately and preview z level can be used to go through them. Scars detection method is same as in the Mark Scars module for images.

Volume Data → *Correct Data* → *Estimate Drift* performs detection and removal of drift in each *xy* plane. Drift is detected by cross-correlation which can be applied either to adjacent planes, one by one, which is optimal for larger drifts observed during the image stack acquisition, or an average plane can be used as a reference, which is optimal for small drifts or noisy data. Preview z level can be used to go through the planes, including visualisation of the cropped data after application of drift correction. On top of XYZ drift estimation, the images can be also rotated in given range, searching for the rotation angle which gives the best correlation score.



Estimate drift dialogue showing detected drift and cropped data.

Super-resolution methods

When multiple images of the same surface area are measured and the image stack is stored as Volume data, they can be used for a super-resolution image calculation, e.g. using some hypothesis about sample statistical parameters.

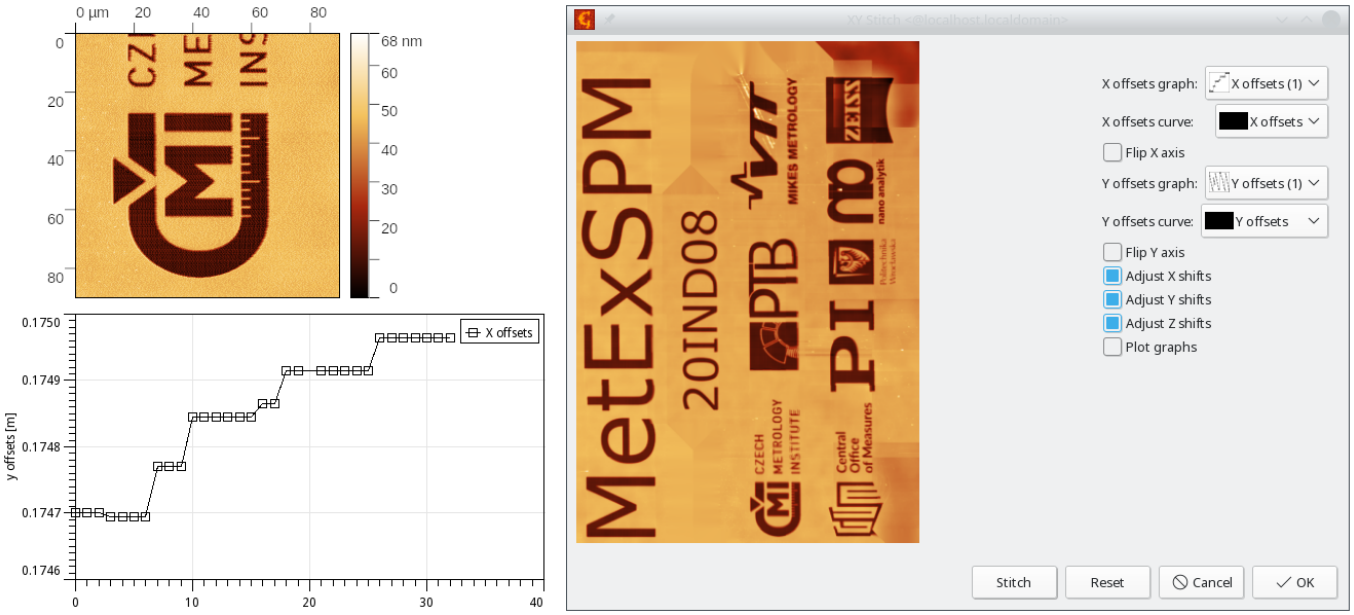
Volume Data → *SPM Modes* → *Localization Merge* performs the localisation algorithm [1]. At every *xy* plane the local peaks are detected, a probability for peak location is expressed and summed across the planes. The method works best on biomolecules or similar objects consisting of resolvable peaks, lying on a flat surface, and can remove the tip convolution impact and provide a high resolution image from series of low resolution ones. An important assumption is that the tip touches slightly different parts of the object at each scan, making the method well suited to soft samples in liquid environment. Note that the individual frames have to be aligned to display the same area of the surface, for this, e.g. the Estimate Drift module can be used.

A single high-speed AFM frame and a localisation microscopy algorithm result, data courtesy of George R. Heath and Simon Scheuring.

Stitching

When the image stack represents different areas on the surface, a merged data representing larger area of the surface can be created. For this an information coming from sample coarse motion stage or from detected image overlaps can be used.

Volume Data → *SPM Modes* → *XY Stitch* performs stitching of a set of images represented as volume data, using the overlaps to minimize mutual offsets and using graphs of X and Y translations between the images. The X and Y translations are in base units (i.e. usually meters) and its X axis is treated as the frame number.



Volume data stitching example, showing one of 32 images taken on MetExSPM project test sample (manufactured by PTB), coarse movement stage data in X axis and the stitched data preview in the module GUI.

References

[1] G.R Heath, E. Kots, J.L. Robertson, et al.: Localization atomic force microscopy. Nature 594 (2021) 385–390, 10.1038/-s41586-021-03551-x

4.26 XYZ Data Processing

The XYZ data processing possibilities are currently rather limited. Most analysis has to be done after their conversion to an image – the basic type of data for which Gwyddion offers plenty of functions.

Basic XYZ data operations currently include merging of two point sets, available as *XYZ Data* → *Merge*. Merging can avoid creation of points at exactly the same lateral coordinates. Instead, their values are averaged – this is enabled by *Average coincident points*.

Rasterization 

XYZ Data → *Rasterize*

Gwyddion excels at working with data sampled in a regular grid, i.e. image data. To apply its data processing functions to irregular XYZ data, such data must be interpolated to a regular grid. In other words, rasterized.

Several interpolation methods are available, controlled bu the *Interpolation type* option:

Round This interpolation is analogous to the Round (nearest neighbour) interpolation for regular grids. The interpolated value in a point in the plane equals to the value of the nearest point in the XYZ point set. This means the Voronoi tessellation is performed and each Voronoi cell is "filled" with the value of the nearest point.

Linear This interpolation is analogous to the Linear interpolation for regular grids. The interpolated value in a point is calculated from the three vertices of the Delaunay triangulation triangle containing the point. As the tree vertices uniquely determine a plane in the space, the value in the point is defined by this plane.

Field The value in a point is the weighted average of all the XYZ point set where the weight is proportional to the inverse fourth power of the mutual distance. Since all XYZ data points are considered for the calculation of each interpolated point this method can be very slow.

Average Ad hoc method obtaining pixel values using a combination of simple averaging and propagation. In dense locations where many XYZ points fall into a single pixel the value assigned to such pixel is a sort of weighted mean value of the points. In sparse locations where relatively large regions contain no XYZ data points the pixel value is propagated from close pixels that contain some XYZ data points. The main feature of this interpolation type is that it is always fast while often producing quite an acceptable result.

The first two interpolation types are based on Voronoi tessellation and Delaunay triangulation that is not well-defined for point sets where more than two points lie on a line or more than three lie on a circle. If this happens the triangulation can fail and an error message is displayed.

The values outside the convex hull of the XYZ point set in the plane are influenced by *Exterior type*:

Border The point set is not amended in any manner and the values on the convex hull simply extend to the infinity.

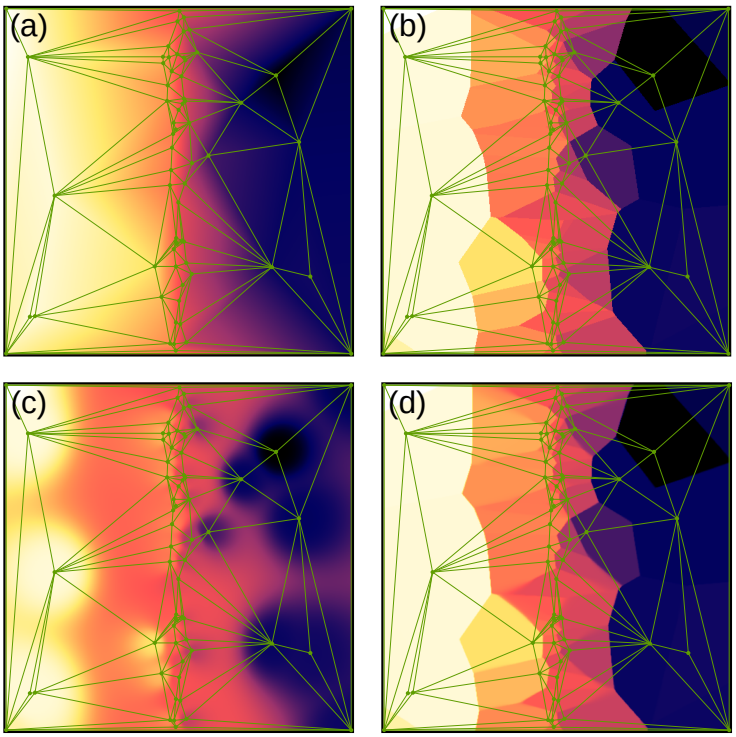
Mirror The point set is amended by points "reflected" about the bounding box sides.

Periodic The point set is amended by periodically repeated points from around the opposite side of bounding box.

The horizontal and vertical pixel dimensions of the resulting image are specified as *Horizontal size* and *Vertical size* in the *Resolution* section.

Often resulting image should have square pixels. This can be achieved by pressing button *Make Pixels Square*. The function does not try to enforce square pixels during the resolution and range editing to avoid changing values that you in fact want to keep. So you always need to make the pixels square explicitly.

It is possible to rasterize only a portion of the XYZ data and also render a part of their exterior. The region to render to image is controlled by the ranges specified in the *Physical Dimensions* section of the dialogue. Button *Reset Ranges* sets the region to a rectangle containing the XYZ data completely. It is also possible to select the region to render on the preview, of course provided that it lies inside the currently displayed region.



Delaunay triangulation displayed on top of (a) Linear, (b) Round, (c) Field and (d) Average interpolations of an irregular set of points. Since the points are sparse, Average interpolation is quite similar to Round, only a bit blurred.

If the XYZ data represent an image, i.e. the points form a regular grid oriented along the Cartesian axes with each point in a rectangular region present exactly once, the rasterization function can directly produce the corresponding image. This is done using button *Create Image Directly* that appears at the top of the dialogue in this case.

Levelling

XYZ Data → *Fix Zero*

XYZ Data → *Plane Level*

The basic *Fix Zero* function is exactly the same as Fix Zero for image data. It shifts all values to make the minimum equal to zero.

Plane Level removes the mean plane. It offers two methods. *Subtraction* is completely analogous to Plane Level for image data and consists in simple subtraction of the mean plane which is found using ordinary least-squares method.

Rotation removes the mean plane by true rotation of the point cloud, an operation exactly possibly only with XYZ data (the corresponding Level Rotate for images is approximate). Furthermore, the function ensures that the mean plane is horizontal after rotation, which corresponds to removal of mean plane in the total least-squares sense. Of course, since rotation mixes lateral coordinates and data values, it is only possible when z is the same physical quantity as x and y (presumably length).

Rotation changes the x and y point coordinates, potentially making the XYZ data incompatible with other data in the file. By enabling *Update X and Y of all compatible data* you can update the lateral coordinates of all other XYZ data sets in the file to match the lateral coordinates in the current one after rotation. The z values of the other data are, of course, kept intact.

Fit Shape

XYZ Data → *Fit Shape*

This function is also available for images. See Fit Shape for images for differences between the image and XYZ variants.

Least-squares fitting of geometrical shapes and other predefined functions to the entire data can serve diverse purposes: removal of overall form such as spherical surface, measurement of geometrical parameters or creation of idealised data matching imperfect real topography. It is also possible to use the module for generation of artificial data if nothing is fitted and all the geometrical parameters are entered explicitly, altough this still requires providing input data that define the XY points.

The basic usage requires choosing the type of shape to fit, selected as *Function type* (see the description of fitting shapes), and pressing the following three buttons in given order:

Estimate Automatic initial estimate of the parameters. Usually the initial estimate is sufficient to proceed with fitting. When it is too off it may be necessary to adjust some of the parameters manually to help the least-squares fitting find the correct minimum. For some functions the estimation uses a randomly chosen subset of the input data to avoid taking too much time. Hence it is non-deterministic and pressing the button again can result in a somewhat different initial parameter estimate.

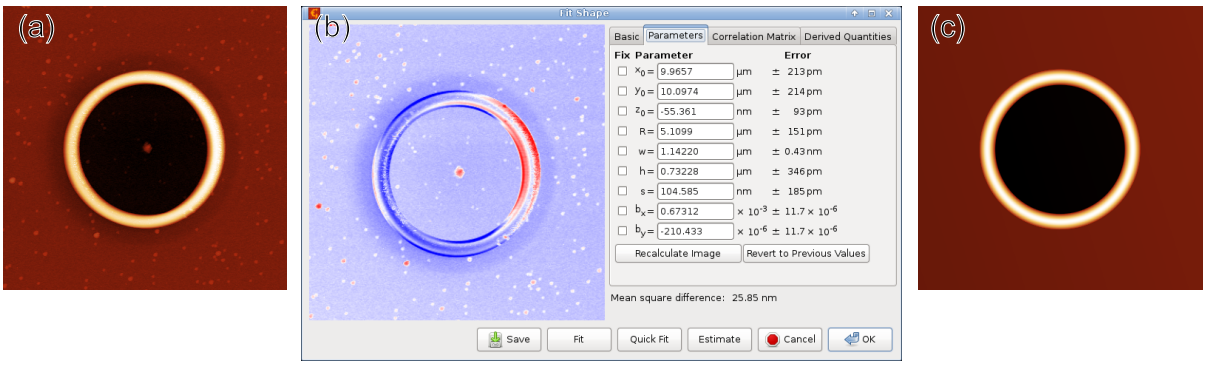
Quick Fit Least-squares fit using a randomly chosen subset of the input data. For large input data it is much faster than the full fit but in most cases it will converge to parameters quite close to final. It allows checking quickly whether the least-squares method is going to converge to the expected minimum – and simultaneously getting the parameters values closer to this minimum for the subsequent full fit.

Fit Full least-squares fit using the entire data. This can take some time, especially for large input data and functions that are slow to evaluate.

The residual mean square difference per input point is displayed below the tabs as *Mean square difference*. It is recalculated whenever the preview is recalculated, so not only after fitting but also after estimation and manual adjustments.

The basic controls in the first tab also allow selecting the output from the fitting, which can be either the fitted shape, the difference between input data and fitted shape, or both.

The preview image can show either the input data, the fitted shape or the difference between the two – which is usually the most useful option. The difference can be displayed with an adapted color map in which red color means the input data are above the fitted shape and blue color means the input data are below the fit. This is enabled with *Show differences with adapted color map*.



Shape fitting example: (a) original topographial data of a ring structure on the surface of a thin film, (b) fitting dialog with parameters on the right side and difference between data and fit displayed in the left part, (c) resulting fitted shape.

The *Parameters* tab displays the values of all fitting parameters and their errors and allows their precise control. Each parameter can be free or fixed. Fixed parameters are not touched by any estimation or fitting function; they are kept at the value you entered. When you change parameter value manually the preview is not recalculated automatically – press *Recalculate Image* to update it. Button *Revert to Previous Values* allows returning to the previous parameter set. For estimation, fitting or manual manipulation this means the set of parameter values before the operation modified them. For reverting it means the parameter set before reverting – so pressing the button repeatedly alternates between the last two parameter sets.

Tab *Correlation Matrix* displays, after a successful fit, the correlation matrix. Correlation coefficients that are very close to unity (in absolute value) are highlighted.

Finally, tab *Derived Quantities* displays various useful values calculated from fitting parameters. Some functions have no derived quantities, some have several. Most derived quantities represent parameters that you may be interested in more than in the actual fitting parameters but that are unsuitable as fitting parameters due to problems with numerical stability. The derived quantities are displayed with error estimates, calculated using law of error propagation (including parameter correlations).

A typical example is the curvature of spherical surface versus its radius of curvature. While the radius is more common it goes from infinity to negative infinity when the surface changes between convex and concave (i.e. is very close to flat), making it unsuitable as a fitting parameter. In contrast, curvature is zero for a flat surface. Therefore, curvature is used as the fitting parameter and the radius of curvature is displayed as a derived quantity.

4.27 Curve Maps Data Processing

Curve maps are formed by bundles of curves measured on a regular grid, so the data processing tasks are either connected to extracting the individual curve data, or to automatically processing the curves in every point to create a map of some evaluated quantity.

Basic curve map operations include simple tasks available as a set of menu entries *Curve Maps* → *Basic operations*. This includes functions for cropping, flipping, rotating curve maps, as well as for changing their physical scale, having the same meaning as for the 2D data. Such functions are useful for any type of curve maps. In contrast to this, many functions are suitable only for some particular curve map types, namely for maps of force-distance curves that many SPMs can produce.

Note that at present the curve map functions typically do not store undo data as the size of a typical curve map is already very large.

Crop 

Curve Maps → *Basic operations* → *Crop*

The module cuts the curve map in lateral directions that can be entered either in pixels or in physical dimensions. With *Keep lateral offsets* option enabled, the top left corner coordinates of the resulting image correspond to the top left corner of the selection, otherwise the top left corner coordinates are set to $(0, 0)$.

Flip 

Flip the data horizontally (i.e. about the vertical axis), vertically (i.e. about the horizontal axis) or in both axes with *Curve Maps* → *Basic Operations* → *Flip Horizontally*, *Flip Vertically* or *Flip Both Axes*, respectively.

Rotation by multiples of 90° 

Rotation of data by 90 degrees is done using one of the basic rotation functions: *Curve Maps* → *Basic Operations* → *Rotate Clockwise*, *Rotate Anticlockwise* .

Dimensions and Units 

Curve Maps → *Basic operations* → *Dimensions and Units*

The module changes physical dimensions of curve maps. This is useful for the correction of raw data that have been imported with wrong physical scales or as a simple manual recalibration of dimensions and values. Apart of different methods for determining the scaling factors (matching pixel size of another data set, setting the scale directly or setting the physical dimensions), the module can be also used to set the lateral offsets. Finally, it can be also used to set the lateral scale units if this was not done while loading the file.

Remove Segments

Curve Maps → *Basic operations* → *Remove Segments*

The module removes any curve map segmentation. Segments are formed by marks on datasets in each pixel of the curve map and can be used e.g. for splitting the force-distance curve into an approach and a retract part in the force-distance data processing modules. Using this module all such marks are deleted.

Align

Curve Maps → *Align...*

The module performs alignment of the curves in the 'z' direction, where 'z' can mean height, but also any other variable on which the spectra are dependent, like voltage in a IV characteristics. A typical use is aligning the Force-Distance curves measured on a sample with large topography variations, obtaining the topography and shifting all the curves to start at the same z level.

Extract Curves

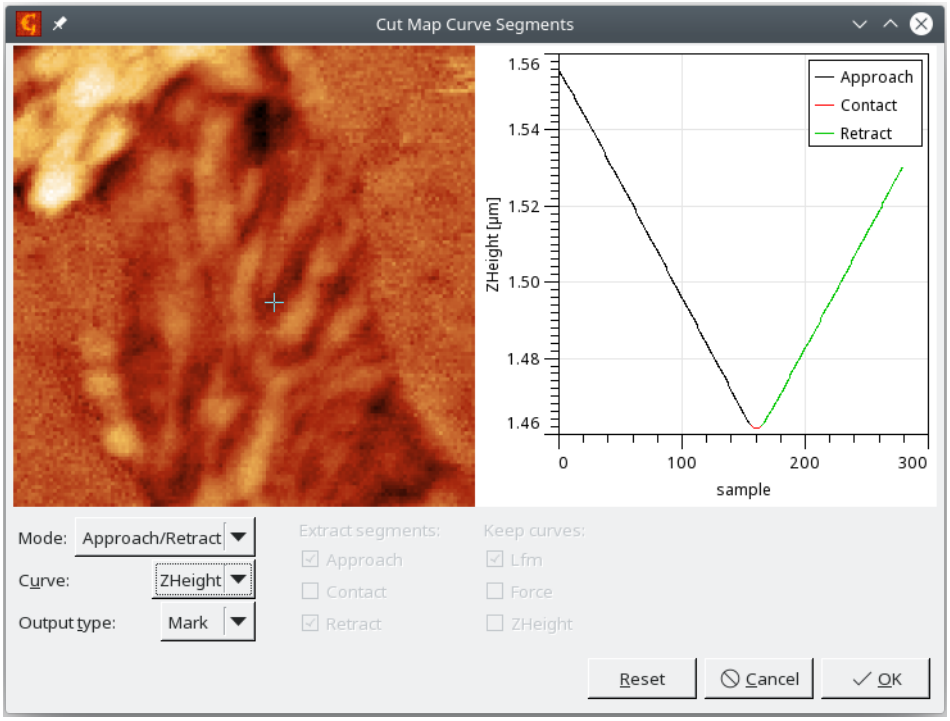
Curve Maps → Extract Curves

The module extracts individual curves as a graph, displayed in the graph window. As there is no hierarchy in the data forming the curve map, it is necessary to select *Abscissa* (the independent variable) and *Ordinate* (the dependent variable) to do so. Typically, for a force-distance curve, the *Abscissa* will be something related to height or probe-sample separation and *Ordinate* will be something related to force. Multiple curves can be extracted by clicking on the *Extract Multiple* option.

Cut to Segments

Curve Maps → Cut to Segments

The module can be used to either create segments on the curve map or to extract these segments. Segments are the key prerequisite for some of the curve map analysis functions. For example, they can represent the approach and retract part of force-distance data. The module uses a *Curve* selected by the user and applies some method (selected by *Mode* parameter) to detect how to split (or mark) the data. For the mentioned case of force-distance measurements the most logical split would be based on the height, segmenting the curve to an approach, hold and retract part as shown here.



Screenshot of Cut to Segments module, showing typical segmentation settings for force-distance data.

The *Output type* can be set to mark the segments only, which is the most typical application. Here, the curve map is not altered, only positions of segments for each individual pixel are added. This is the mode which should be selected to prepare data for curve map processing modules that use segmented data. The second option is to extract some segments, including the selection of which curves should be kept. This can be also used to shrink the data size significantly, keeping only what is needed.

4.28 Force Distance Curve Maps

An important type of curve maps is based on force-distance data acquired on a regular grid. All major SPM producers offer some regime for the acquisition of such data while the main goal is to map the mechanical properties of the sample. In Gwyddion, these data can be represented as curve maps.

In this section we describe the modules that can be used to handle force-distance curve maps and also give an example of a toolchain for obtaining quantitative nanomechanical results from measured data.

Note that for many of the discussed modules it is important to use curve maps that are already segmented. This means that the approach and retract part of the cuve is marked, either by coming already marked from the instrument, or being marked in Gwyddion using the segmentation module. Also note that the modules assume that the dependent variable is the probe-sample distance - being negative for the indentation part of the curve and positive for the part of the curve when the probe is not touching the surface, as seen on screenshots in this section.

Remove Polynomial Background

Curve Maps → *Remove Polynomial Background*

The module fits a requested part of the force-distance curve by a polynomial of some *Degree* and then subtracts it from the whole curve. This is done for every pixel and can be used to remove some unwanted background from the zero line (part of the data where the probe is far from the surface). Parameters *Abscissa* and *Ordinate* are used to set the dependent and independent variable, which will usually be something probe-sample distance dependent for *Abscissa* and something force dependent for *Ordinate*. Fitting can be restricted to some particular *Segment*. The fitting range is entered in percent, so it is recalculated for every pixel based on the curve length, regardless of the absolute offset of the curves.

Remove Sine Background

Curve Maps → *Remove Sine Background*

The module fits a requested part of the force-distance curve by a sine function and then subtracts it from the whole curve. This is done for every pixel and can be used to remove some unwanted background related to interference effects from the zero line (part of the data where probe is far from surface). Parameters *Abscissa* and *Ordinate* are used to set the dependent and independent variable, which will usually be something probe-sample distance dependent for *Abscissa* and something force dependent for *Ordinate*. Fitting can be restricted to some particular *Segment*. The fitting range is entered in percent, so it is recalculated for every pixel based on the curve length, regardless of the absolute offset of the curves.

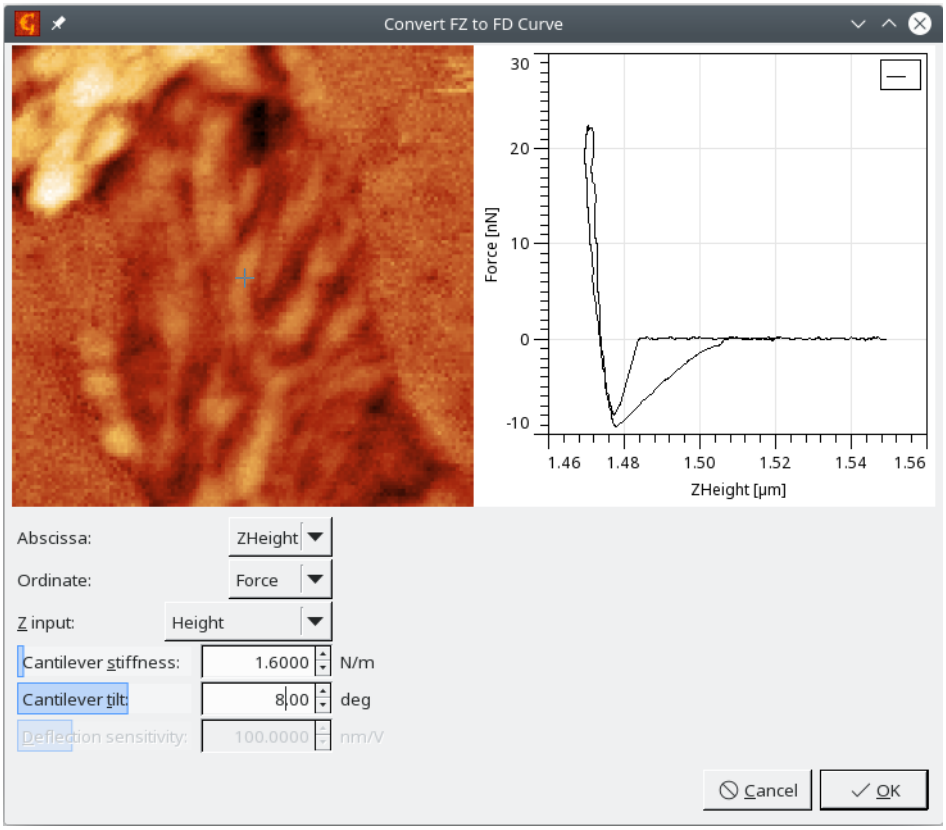
FZ to FD Curve

Curve Maps → *FZ to FD Curve*

The module performs the conversion of the independent variable in force-distance curves from z-piezo related information (often called z-piezo, height, etc.) to the separation between probe and sample, here called distance. This takes into account the impact of finite cantilever stiffness, which deflects during the force-distance data acquisition. That's why *Cantilever stiffness* has to be entered. This value can be sometimes found in the metadata, but sometimes might come from another experiment. The lower the cantilever stiffness, the bigger its impact on the curve. A very low value can even change the curve slope to be unrealistic.

Data coming from different instruments can use different orientations of the z axis and the option *Z input* can be used to reflect it. The goal is to get a curve that has its force maximum at the left side, i.e. the x-axis on the graph is the probe-sample displacement, which is negative when the probe is pressed into the sample.

Effective stiffening of the cantilever related to the angle between cantilever and sample surface can be added using the parameter *Cantilever tilt*. If the data units for force are Volts, the parameter *Deflection sensitivity* is automatically activated and can be used to convert the data to force values.



Screenshot of the FZ to FD module.

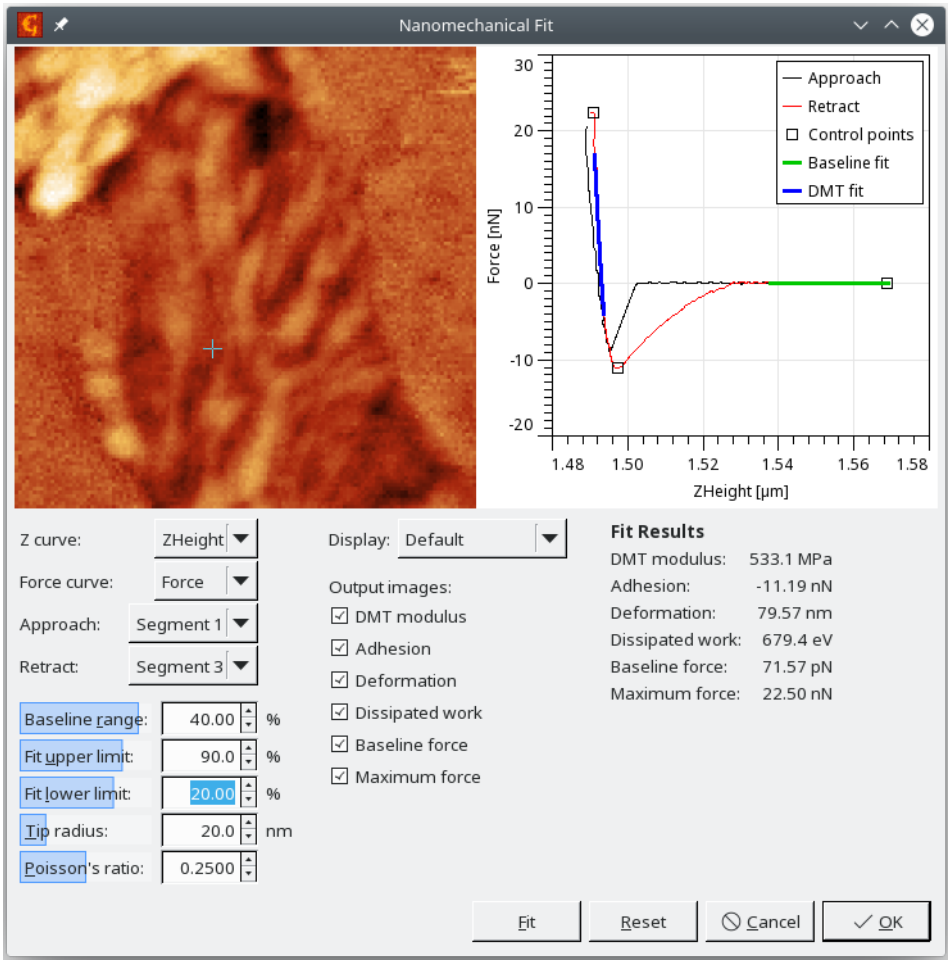
Nanomechanical Fit

Curve Maps → Nanomechanical Fit

The module performs the automated calculation of simple mechanical properties from force-distance curve maps. The user can enter the ranges in which the baseline (zero line) and the retract part fit will be evaluated and also needs to enter the *Tip radius* and *Poisson's ratio*. The right curves need to be picked from the dataset in every pixel and also the right segments need to be selected. These settings are however remembered, so if similar data are processed, there should be no need for setting them.

The adhesion is evaluated as the force between the minimum value on the retract curve and the mean value on the selected part of baseline. Deformation corresponds to the length of the repulsive part of the approach curve, i.e. it is the distance between the position when the force reaches zero after jumping into contact and the position of the maximum force. DMT modulus is determined by fitting a selected part of the retract curve. Dissipation work is calculated as the difference of the integrated approach and retract curves.

The user can select which of these values will be output as arrays and let the module evaluate the data in every pixel using the *Fit* button.



Screenshot of the Nanomechanical Fit module.

At the end a set of data fields is produced corresponding to the spatial distribution of different results across the surface.

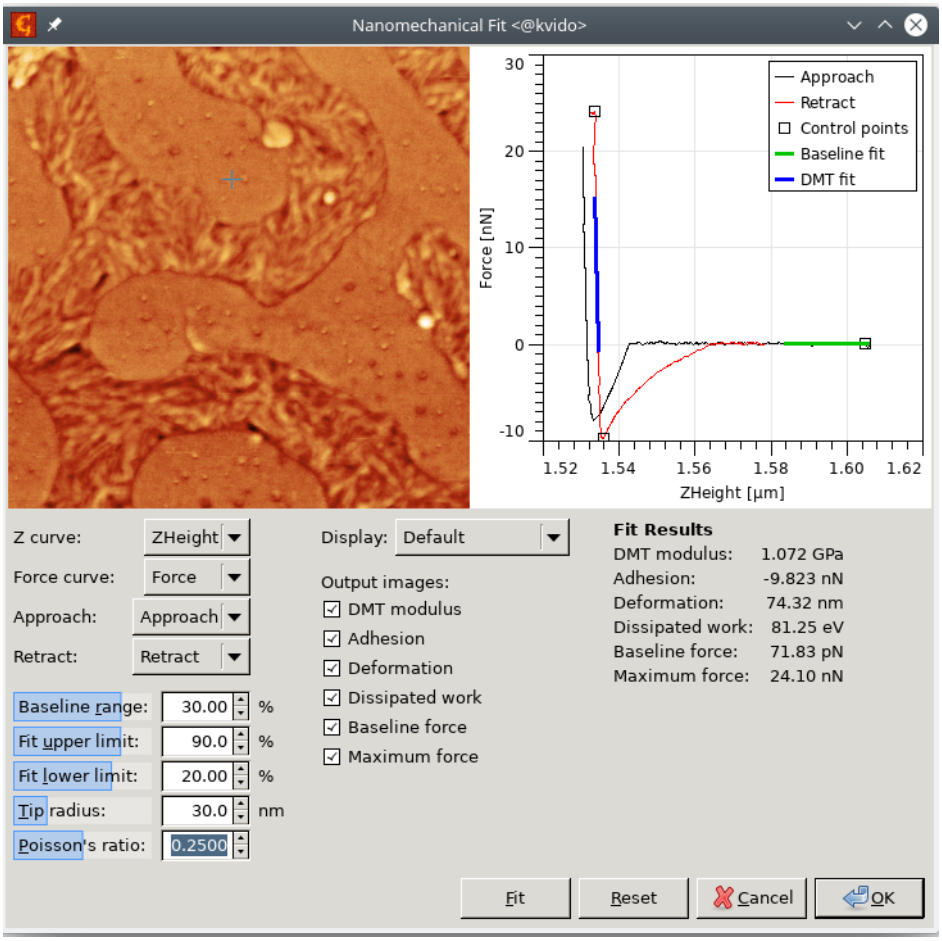
FD curve analysis toolchain example

To illustrate the whole procedure of obtaining nanomechanical results from force-distance curve maps we show here a detailed example how the different modules can be combined.

The data used for this example were produced by Park Systems in the PinPoint regime, using an SD-R30-FM cantilever. The sample is formed by two blended polymers. The input file is available for download. As curve maps are always quite memory demanding, we strongly encourage you to use ithe 64 bit version of the operating system for handling them.

To go step by step and get the same results as provided in the processed file, the following procedure has to be done:

1. Right click on the data, open the metadata browser and write down the forceConstant value, which is the cantilever stiffness for this particular experiment.
2. Add segments to the data using Cut to Segments module. Use Approach/Retract segmentation mode, and ZHeight curve. Output type should be 'Mark'.
3. Use FZ to FD curve module to remove the impact of cantilever stiffness from the data, i.e. to convert the cantilever base height to probe-sample distance. We will use our cantilever stiffness here. Note that it is different from the manufacturer's specifications, so it was probably measured during the experiment, which is an important step towards quantitative measurements. Choose ZHeight for Abscissa, Force for Ordinate and Height as the Z Input.
4. Calculate the results using the Nanomechanical Fit module, using the settings shown below:



Screenshot of the Nanomechanical Fit module applied on the example data.

Within the module you can recalculate and preview the individual channels. After clicking on the OK button, you should obtain a set of images representing the spatial distribution of different nanomechanical parameters.

4.29 Synthetic Surfaces

Beside functions for analysis of measured data, Gwyddion provides several generators of artificial surfaces and measurement artefacts that can be used for testing or simulations also outside Gwyddion [1].

All the surface generators share a certain set of parameters, determining the dimensions and scales of the created surface and the random number generator controls. These parameters are described below, the parameters specific to each generator are described in the corresponding subsections.

Image parameters:

Horizontal, Vertical size The horizontal and vertical resolution of the generated surface in pixels.

Square image This option, when enabled, forces the horizontal and vertical resolution to be identical.

Width, Height The horizontal and vertical physical dimensions of the generated surface in selected units. Note square pixels are assumed so, changing one causes the other to be recalculated.

Dimension, Value units Units of the lateral dimensions (*Width*, *Height*) and of the values (heights). The units chosen here also determine the units of non-dimensionless parameters of the individual generators.

Take Dimensions from Current Image Clicking this button fills all the above parameters according to the current image.

Note that while the units of values are updated, the value scale is defined by generator-specific parameters that might not be directly derivable from the statistical properties of the current image. Hence these parameters are not recalculated.

Replace the current image This option has two effects. First, it causes the dimensions and scales to be automatically set to those of the current image. Second, it makes the generated surface replace the current image instead of creating a new image.

Start from the current image This option has two effects. First, it causes the dimensions and scales to be automatically set to those of the current image. Second, it makes the generator to start from the surface contained in the current image and modify it instead of starting from a flat surface. Note this does not affect whether the result actually goes to the current image or a new image is created.

Random generator controls:

Random seed The random number generator seed. Choosing the same parameters and resolutions and the same random seed leads to the same surface, even on different computers. Different seeds lead to different surfaces with the same overall characteristics given by the generator parameters.

New Replaces the seed with a random number.

Randomize Enabling this option makes the seed to be chosen randomly every time the generator is run. This permits to conveniently re-run the generator with a new seed simply by pressing **Ctrl-F** (see keyboard shortcuts).

Update/preview controls:

Instant updates If the checkbox is enabled the surface is recalculated whenever an input parameter changes. This option is available in generators which are relatively fast.

Progressive preview If the checkbox is enabled the preview shows the evolution of the surface during the simulation. This option is available in slow generators involving iterative physical simulations. Enabling the progressive generally makes the simulation slightly slower as snapshots need to be taken and displayed.

Spectral 

The spectral synthesis module creates randomly rough surfaces by constructing the Fourier transform of the surface according to specified parameters and then performing the inverse Fourier transform to obtain the real surface. The generated surfaces are periodic (i.e. perfectly tilable).

The Fourier image parameters define the shape of the PSDF, i.e. the Fourier coefficient modulus, the phases are chosen randomly. At present, all generated surfaces are isotropic, i.e. the PSDF is radially symmetric.

RMS The root mean square value of the heights (or of the differences from the mean plane which, however, always is the $z = 0$ plane). Button *Like Current Image* sets the RMS value to that of the current image.

Minimum, maximum frequency The minimum and maximum spatial frequency. Increasing the minimum frequency leads to "flattening" of the image, i.e. to removal of large features. Decreasing the maximum frequency limits the sharpness of the features.

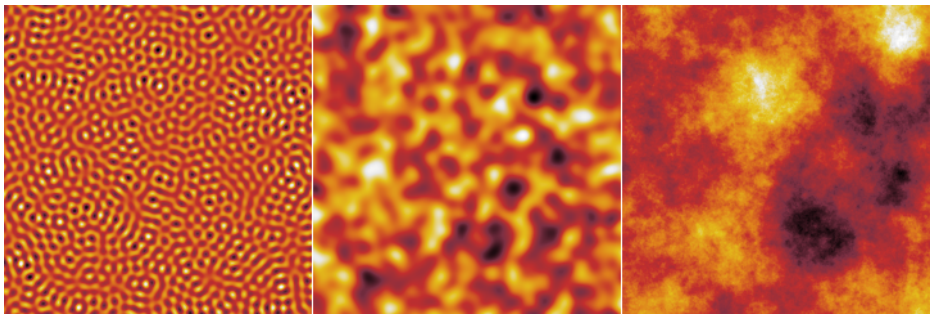
Enable Gaussian multiplier Enables the multiplication of the Fourier coefficients by a Gaussian function that in the real space corresponds to the convolution with a Gaussian.

Enable Lorentzian multiplier Enables the multiplication of the Fourier coefficients by a function proportional to $1/(1 + k^2T^2)^{3/4}$, where T is the autocorrelation length. So, the factor itself is not actually Lorentzian but it corresponds to Lorentzian one-dimensional power spectrum density which in turn corresponds to exponential autocorrelation function (see section Statistical Analysis for the discussion of autocorrelation functions). This factor decreases relatively slowly so the finite resolution plays usually a larger role than in the case of Gaussian.

Autocorrelation length The autocorrelation length of the Gaussian or Lorentzian factors (see section Statistical Analysis for the discussion of autocorrelation functions).

Enable power multiplier Enables multiplication of Fourier coefficients by factor proportional to $1/k^p$, where k is the spatial frequency and p is the power. This permits to generate various fractal surfaces.

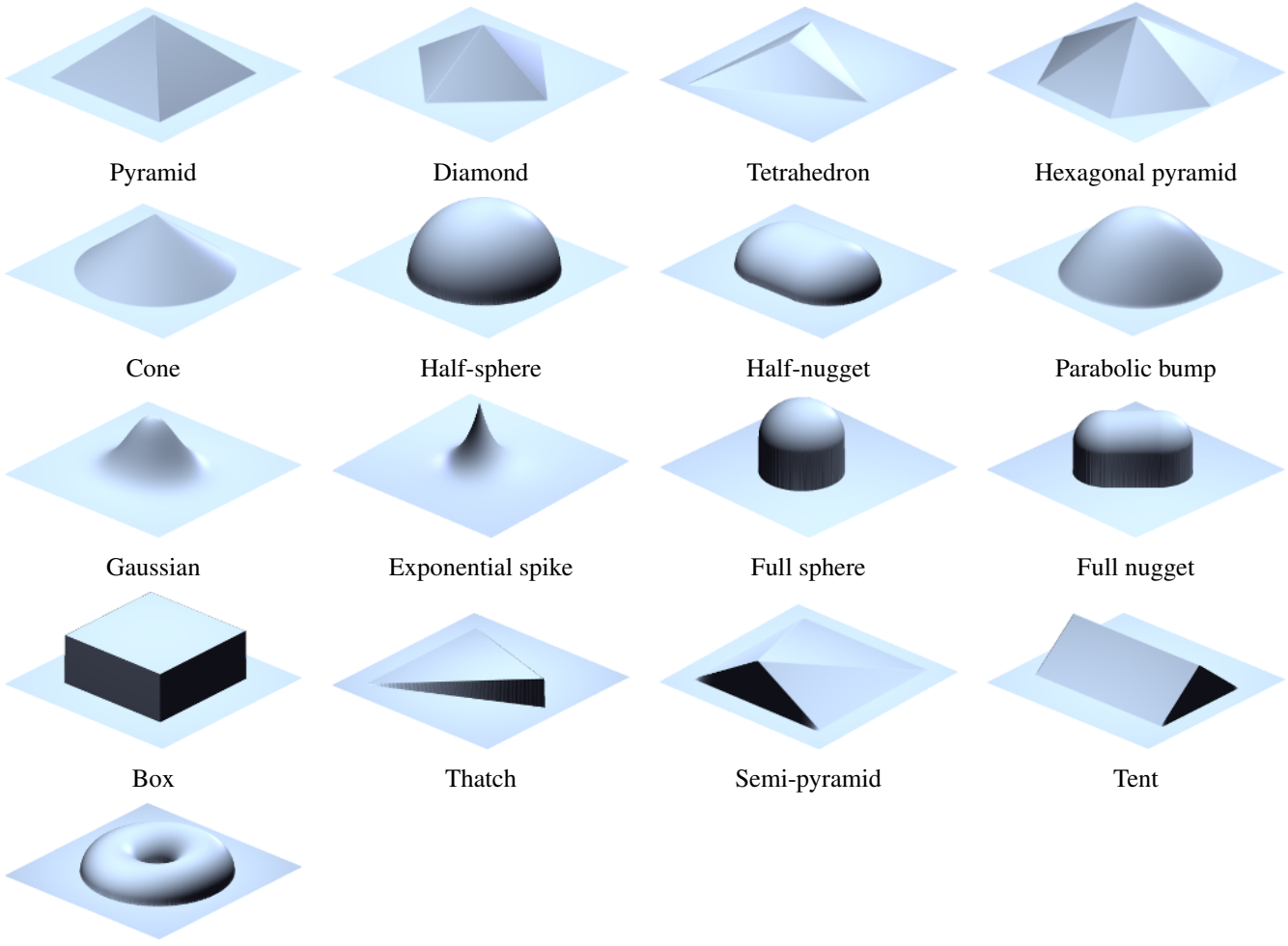
Power The power p.



Artificial surfaces generated by spectral synthesis: a narrow range of spatial frequencies (left), Gaussian random surface (centre) and a fractal surface generated with power multiplier and p equal to 1.5 (right).

Objects 

The object placement method permits to create random surfaces composed of features of a specific shape. The algorithm is simple: the given number of objects is placed on random positions at the surface. For each object placed, the new heights are changed to $\max(z, z_0 + h)$, where z is the current height at a specific pixel, h is the height of the object at this pixel (assuming a zero basis) and z_0 is the current minimum height over the basis of the object being placed. The algorithm considers the horizontal plane to be filled with identical copies of the surface, hence, the generated surfaces are also periodic (i.e. perfectly tilable).



Parameters and options in the *Shape* tab control the shape of individual features:

Shape The shape (type) of placed objects. At present the possibilities include half-spheres, boxes, various pyramids, bumps, spikes and a few more weird shapes. The difference between half-sphere and full sphere is in the height. The full sphere includes also the height of the lower half. The same applies to nuggets.

Size The lateral object size, usually the side of a containing square.

Aspect Ratio The ratio between the x and y dimensions of an object – with respect to some default proportions.

Changing the aspect ratio does not always imply mere geometrical scaling, e.g. objects called nuggets change between half-spheres and rods when the ratio is changed.

Height A quantity proportional to the height of the object, normally the height of the highest point.

Enabling *Scales with size* makes unperturbed heights to scale proportionally to the sizes of individual objects. Otherwise the height is independent on size. Proportional scaling is useful with non-zero size variance to keep the shape of all objects identical (as opposed to elongating and flattening them vertically). If all objects have the same size the option has no effect.

Button *Like Current Image* sets the height value to a value based on the RMS of the current image.

Truncate The shapes can be truncated at a certain height, enabling creation of truncated cones, pyramids, etc. The truncation height is given as a proportion to the total object height. Unity means the shape is not truncated, zero would mean complete removal of the object.

Each parameter can be randomized for individual objects, this is controlled by *Variance*. For multiplicative quantities (all except orientation and truncation), the distribution is log-normal with the RMS value of the logarithmed quantity given by *Variance*.

Parameters and options in the *Placement* tab control where and how the surface is modified using the generated shapes:

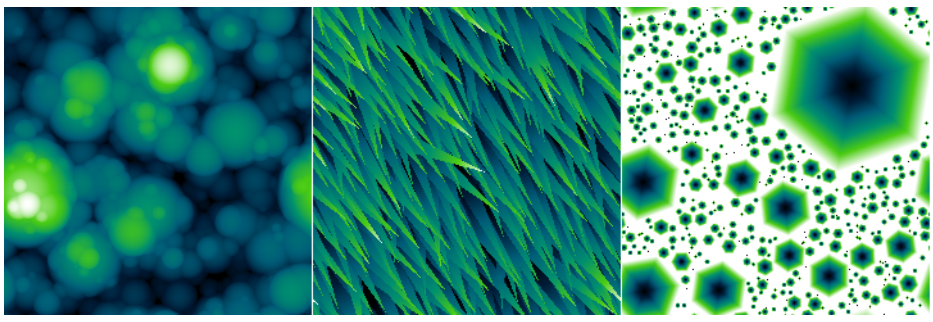
Coverage The average number of times an object covers a pixel on the image. Coverage value of 1 means the surface would be exactly once covered by the objects assuming that they covered it uniformly and did not overlap. Values larger than 1 mean more layers of objects – and slower image generation.

Feature sign The direction in which the surface is changed by adding objects. Positive and negative modifications correspond forming hills and pits adding objects, respectively. Zero value of the parameter means hills and pits are chosen randomly. Positive values mean hills are more likely, while the maximum value 1 means only hills. The same holds for pits and negative values.

Columnarity For columnarity of zero the local surface modification works as described in the introduction. For columnarity of one (the maximum), the objects are considered to have also a lower part, which is a perfect mirror of the (visible) upper part. Once the lower object part touches any point on the surface, it sticks to it. This determines the final height. The range of heights grows quckly in this case and the volume is rather porous (not representable with a height field, of course). Values between 0 and 1 mean the object height is interpolated between the two extremes.

Avoid stacking When this option is enabled, at most one object may be placed on any pixel of the surface. Hence, they never overlap. The *Coverage* option then does not determine how many objects are actually placed, but only how many the generator tries. Once there is no free space remaning where an object of given size could be added, increasing the coverage has no effect (beside slowing down the generation).

Orientation The rotation of objects with respect to some base orientation, measured anticlockwise. The orientation can be also randomised using the corresponding *Variance*.



Artificial surfaces generated by object placement: spheres of varied size (left), narrow thatches of varied direction (centre), negative non-overlapping hexagonal pyramids of varying size (right).

Particles 

Particle depositions performs a dynamical simulation of interacting spherical particles falling onto a solid surface. The particles do not fall all at once. Instead, they are dropped sequentially among the particles already lying on the surface and left to relax.

Particle radius Radius of the spherical particles in real units. The height and lateral dimensions must be the same physical quantity so there is only one dimensional parameter.

Distribution width Standard deviation of radius distribution (Gaussian).

Coverage How much of the surface would be covered if it was covered uniformly by the deposited particles. If the simulation is unable to place as many particles as requested, usually due to too short time, an error message is displayed.

Relaxation steps Simulation length, measured in the number of iterations.

The rod deposition method is based on the same dynamic simulation, but for elongated particles. The particles are in fact formed by sphere triplets, nevertheless this still allows a reasonable range of particle aspect ratios. The simulation also has several more interaction parameters.

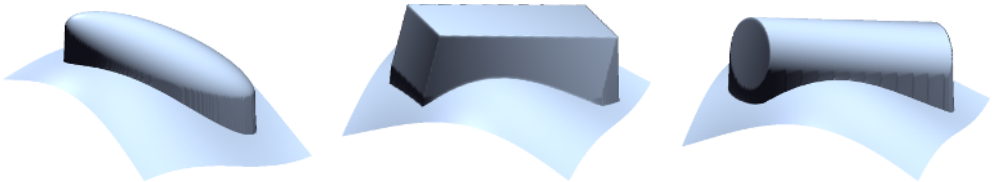
Pile Up 

The Objects artificial surface generator is rather fast but at the expense of avoiding any kind of 3D geometry and just moulding the surface using a few simple rules. Particle and rod depositions lie near the other end of the speed-realism spectrum and perform actual dynamical simulation of interacting particles falling onto the surface. This module represents a middle ground, as it forms the surface from real 3D solids, but still using only simple rules for what happens to them when they hit the surface, retaining a good surface generation speed.

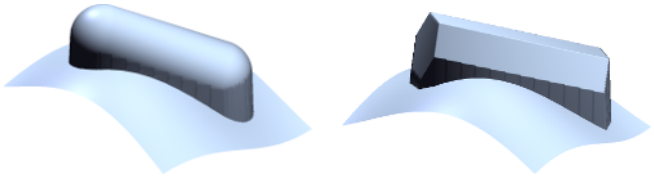
The solids start falling in the horizontal position. When they hit the surface, a local mean plane around in the contact area is found. The shape is then rotated to lie flat on this local plane. Its final height is determined as detailed in the description of *Columnarity* below and then it just sticks in this position. There is no further sliding, toppling or other dynamic settling process.

Several generator parameters have exactly the same meaning as in Objects. This includes *Coverage*, *Avoid stacking* and *Orientation*. The others are explained below:

Shape The shape (type) of placed objects. At present the possibilities include various elongated shapes.



Ellipsoid Bar Cylinder



Nugget Hexagonal rod

Width The shapes can be elongated along the x axis (as controlled by *Aspect ratio*). Their dimension in the orthogonal direction is given by width. The cross section is symmetrical. Therefore, the width determines both the horizontal width and height.

Aspect ratio The ratio between the x and y dimensions, in other words, the elongation. It is always at least unity; the shapes are never squashed along the x axis. The aspect ratio distribution can be thus noticeably asymmetrical when the nominal aspect ratio is around unity and the variation is large.

Columnarity In general, this parameter influences how much the shapes tend to form columnar structures (porous, if topographical images could represent it). The default value of zero can be imagined as the lower part of the shape melting upon impact and filling the depression it has fallen into. Note that the volume balance is somewhat approximate and the height at which the object settles may be such that sharp spikes on the surface pierce through it.

The maximum columnarity (1) means that once any part of the shape touches the surface, it sticks to it. For smaller positive values the final height is interpolated between the two heights gien by the two approaches.

If columnarity is negative, the shape is rammed into the surface. For the minimum value (-1) it settles at the lowest height for which at least one point of its entire lower boundary would not be below the original surface. Again, for other negative values the final height is interpolated.

Artificial surfaces generated by piling up geometrical shapes: oriented hexagonal rods on a central ridge (left), jumble of ellipsoid needles (centre), heap of cubes piled up on a central pillar (right).

Noise 

Random uncorrelated point noise is generated independently in each pixel. Several distributions are available.

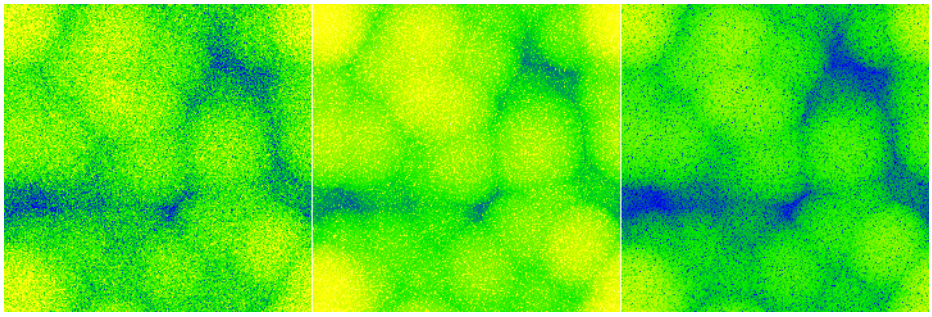
Distribution The distribution of the noise value. The possibilities include Gaussian, exponential, uniform, triangular and salt and pepper distributions.

The salt and pepper distribution works somewhat differently than is usual in computer graphics since the range of values is unbounded. It is a bimodal distribution consiting of two δ-functions. In other words, a constant (given by RMS) is added and/or subtracted from the pixel values.

Direction The noise can be generated as symmetrical or one-sided. The mean value of the distribution of a symmetrical noise is zero, i.e. the mean value of data does not change when a symmetrical noise is added. One-sided noise only increases (if positive) or decreases (if negative) the data values.

RMS Root mean square value of the noise distribution. More precisely, it is the RMS of the corresponding symmetrical distribution in the case the distribution is one-sided.

Density Density is the fraction of values modified. If it is 1 all pixels are modified. When it is smaller some are left intact.



Different artificial noise applied to the same surface: symmetrical Gaussian noise (left); positive exponential noise (centre); negative exponential noise (right). All images have the same false colour scale and all noises have the same RMS.

Line Noise 

Line noise represents noise with non-negligible duration that leads to typical steps or scars (also called strokes) in the direction of the fast scanning axis – or misaligned scan lines. Parameters *Distribution*, *Direction* and *RMS* have the same meaning as in Point noise. Other parameters control the lateral characteristics of the noise.

The following line defect types are available:

- *Steps* represent abrupt changes in the value that continue to the end of the scan (or until another step occurs).
- *Scars* are local changes of the value with a finite duration, i.e. the values return to the original level within the same scan line.
- *Ridges* are similar to scars but on a larger scale: they can span several scan lines.
- *Tilt* perturbs the slope of individual scan lines, as opposed to directly shifting heights.
- *Hum* adds a quasi-periodic disturbance, simulating a type of noise originating from electronic interference.

Steps have the following parameters:

Density Average number of defects per scan line, including any dead time (as determined by parameter *Within line*).

Within line Fraction of the time to scan one line that corresponds to actual data acquisition. The rest of time is a dead time. Value 1 means there is no dead time, i.e. all steps occur within the image. Value 0 means the data acquisition time is negligible to the total line scan time, consequently, steps only occur between lines.

Scanning direction If steps can occur inside a scan line it determines the horizontal scanning direction (assuming the slow axis scan is from top to bottom). For left to right scanning the left part of the scan line is contiguous with the data above and the right part with data below. For right to left scanning it is the opposite. The two directions are illustrated in the descriptions of Block line correction which corrects similar artefacts.

Cumulative For cumulative steps the random step value is always added to the current value offset; for non-cumulative steps the new value offset is directly equal to the random step value.

Scars have the following parameters:

Coverage The fraction of the the image covered by defect if they did not overlap. Since the defect may overlap, coverage value of 1 does not mean the image is covered completely.

Length Scar length in pixels.

Variance Variance of the scar length, see Objects for description of variances.

Ridges have the following parameters:

Density Average number of defects per scan line, including any dead time (as determined by parameter *Within line*).

Within line Fraction of the time to scan one line that corresponds to actual data acquisition. The rest of time is a dead time. Value 1 means there is no dead time, i.e. all value changes occur within the image. Value 0 means the data acquisition time is negligible to the total line scan time, consequently, value changes only occur between lines.

Scanning direction If steps can occur inside a scan line it determines the horizontal scanning direction (assuming the slow axis scan is from top to bottom). For left to right scanning the left part of the scan line is contiguous with the data above and the right part with data below. For right to left scanning it is the opposite. The two directions are illustrated in the descriptions of Block line correction which corrects similar artefacts.

Width Mean duration of the defect, measured in image size. Value 1 means the mean duration will be the entire image scanning time. Small values mean the defects will mostly occupy just one scan line.

Tilt has the following parameters:

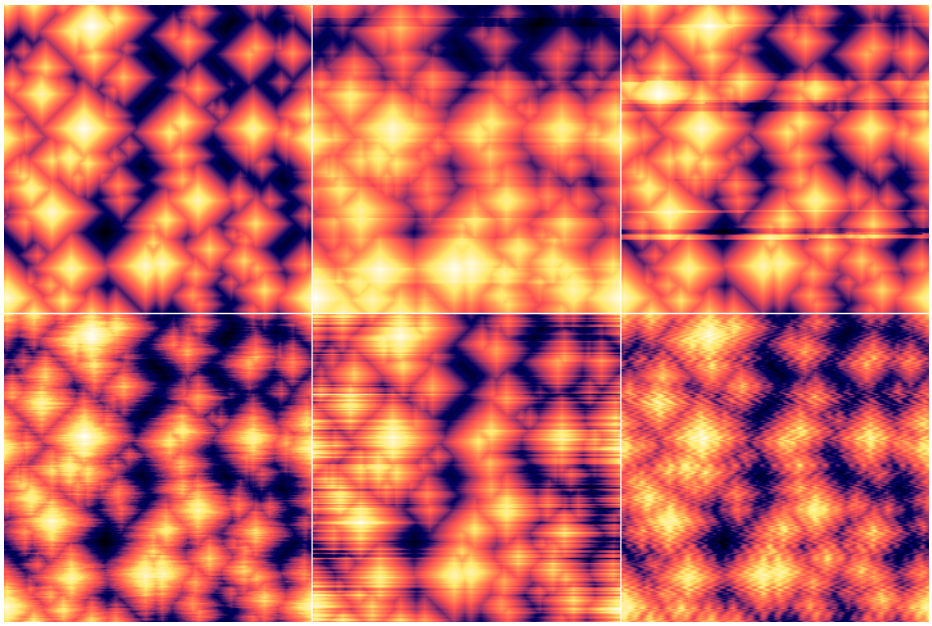
Offset dispersion With zero offsets all scan lines are tilted around the middle. In other words, the central image column remains unperturbed. Increasing the dispersion of the offsets moves the pivot points randomly more and more to left or right.

Hum has the following parameters:

Wavelength Characteristic wavelength of the hum, measured in the image.

Spread Small spread means the noise is very narrow-band and looks more or less like as a pure sine wave. Large values produce a wider spectrum with beats and more distorted profile.

Components The number of random sine wave components that are used to construct the hum in each scan line.



Different types of line noise added to an artificial pyramidal surface. Top row: unmodified surface (left); relatively sparse cumulative steps (centre); relatively sparse ridges (right). Bottom row: scars of mean length of 16 px and high coverage (left); tilt with small offset dispersion (centre); hum with default settings (right).

Pattern

Regular geometrical patterns represent surfaces often encountered in microscopy as standards or testing samples. Currently it can generate the following types of pattern:

Staircase One-dimensional staircase pattern, formed by steps of constant width and height.

Grating One-dimensional pattern, formed by double-sided steps (ridges) constant width and height.

Amphitheatre Superellipse-shaped pattern formed by concentric steps of constant width and height.

Concentric rings Superellipse-shaped pattern formed by concentric double-sided steps (ridges) of constant width and height.

Siemens star Radial pattern formed by alternating high and low wedges.

Holes (rectangular) Two-dimensional regular arrangement of rectangular holes, possibly round.

Pillars Two-dimensional regular arrangement of symmetrical pillars of several possible shapes (circular, square or hexagonal).

Double staircase Two-dimensional regular arrangement of steps of fixed width and height.

Each type of pattern has its own set of geometrical parameters determining the shape and dimensions of various part of the pattern. They are described in the following sections. Most geometrical parametrs have an associated variance control, similar to Object synthesis, which permits making some aspects of the pattern irregular.

The placement of the pattern in the horizontal plane is controlled by parameters in tab *Placement*, common to all pattern types:

Orientation The rotation of the pattern with respect to some base orientation, measured anticlockwise.

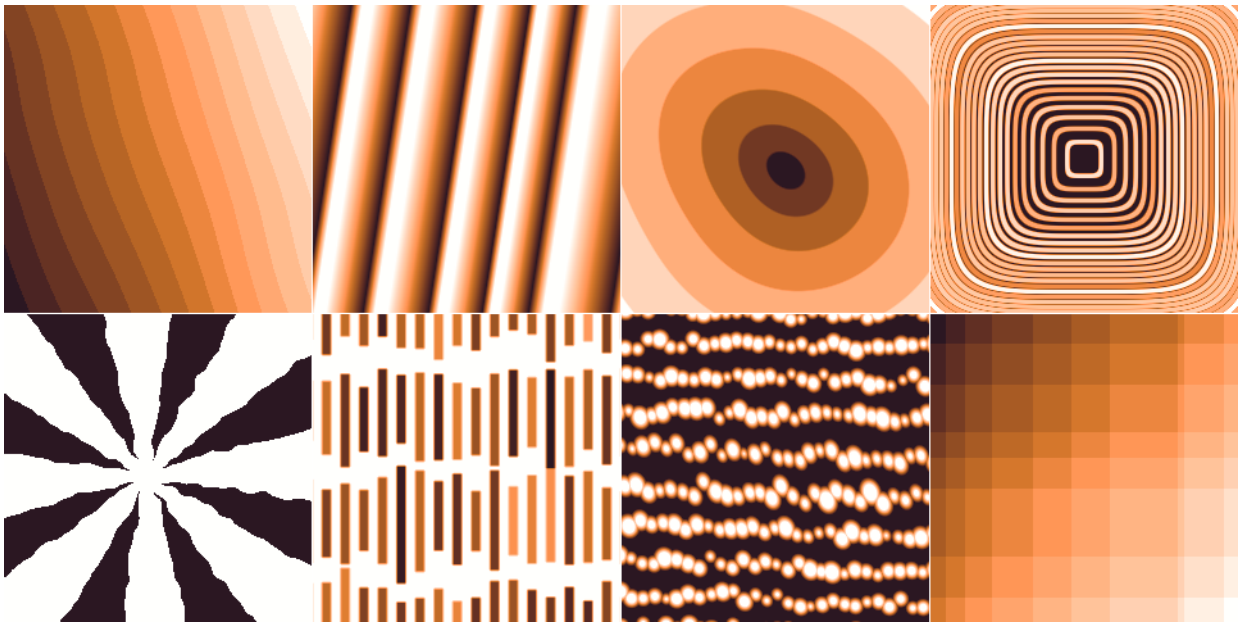
This tab also contains the deformation parameters. While enabling the variation of geometrical parameters makes the generated surface somewhat irregular the shape of its features is maintained. *Deformation* is a complementary method to introduce irregularity, specifically by distorting the pattern in the *xy* plane. It has two parameters:

Amplitude The magnitude of the lateral deformation. It is a relative numerical quantity essentially determining how far the deformation can reach.

Lateral scale The characteristic size of the deformations. It describes not how far the features are moved but how sharply or slowly the deformation itself changes within the horizontal plane.

The deformation is analogous to the two-dimensional Gaussian deformation applied by Displacement field. However, it is applied to coordinates before rendering the pattern, whereas Displacement field distorts already sampled images.

All patterns also have a *Height* parameter, giving the height of features in physical units. For staircase-like patterns that grow continuously it is the height of a single step. For patterns consisting of bumps of holes in a flat base plane it is the height of each individual feature (unless modified by variance).



Artificial pattern surfaces. Top row: sharp steps with long-wavelength deformation, scaled grating with large width variance and wide asymmetrical slopes, a slightly deformed amphitheatre with negative parabolicity, and concentric rings with positive parabolicity and varying height. Bottom row: Siemens star with enlarged upper surface, long rectangular holes with varying vertical dimension and position, rows of slightly deformed circular pillars, and double staircase with irregular step width.

Staircase

Staircase has the following specific parameters:

Terrace width The width of one step, given in pixels but also displayed in physical units. All other lateral dimensions are given as fractions of this base width.

The width includes the slope width and can be seen as a "period" of the pattern.

Position variance Randomisation of the position of step edge. The edges never cross. Even for the maximum randomness, they can at most move to touch the next edge.

Slope width Fraction of width taken by the sloping edge. Zero means sharp edges, one means the pattern will be a flat ramp (without randomness or other modifying parameters).

Scales with width If enabled for the step height, the heights will be adjusted to be proportional to the widths of surrounding terraces, while keeping the average step height as given. This more closely preserves the mean plane of the surface even for randomised terrace width.

Grating

Grating has the following specific parameters:

Period The period of the pattern, given in pixels but also displayed in physical units. All other lateral dimensions are given as fractions of this base width.

Position variance Randomisation of the widths of individual ridge-trough pairs.

Scale features with width If enabled then all lateral dimensions scale with the width of each individual ridge-trough pair for randomised patterns. Otherwise *Top fraction* or *Slope width* refer to the unperturbed base period.

Top fraction Fraction of width taken by the upper surface.

Slope width Fraction of width taken by the sloping edges. Slopes are added to the top surface; they do not replace it. Therefore, zero *Top fraction* and non-zero slope results in a triangular profile.

Asymmetry Distribution of slope width to left and right edges. Minus one means left edges are sloped and right edges are sharp. Plus one means left edges are sharp and right edges are sloped. Zero means symmetrical slopes.

Amphitheatre

Amphitheatre has the following specific parameters:

Superellipse parameter Step contours are superellipses described by $(x^{2/p} + y^{2/p})^{p/2} = \text{const.}$ where p is the parameter. Zero corresponds to squares, the maximum value two corresponds to diamonds, and the default value one corresponds to circles.

Terrace width The width of one step, given in pixels but also displayed in physical units. All other lateral dimensions are given as fractions of this base width.

The width includes the slope width and can be seen as a "period" of the pattern.

Position variance Randomisation of the position of step edge. The edges never cross. Even for the maximum randomness, they can at most move to touch the next edge.

Parabolicity Positive values means that the terrace become narrower with increasing distance from centre and the overall profile becomes close to parabolical. Negative values have the opposite effect, with the overall shape becoming a parabola lying on its side.

Slope width Fraction of width taken by the sloping edge. Zero means sharp edges, one means the pattern will be become a cone (without randomness or other modifying parameters).

Concentric rings

Concentric rings have the following specific parameters:

Superellipse parameter Ring contours are superellipses described by $(x^{2/p} + y^{2/p})^{p/2} = \text{const.}$ where p is the parameter. Zero corresponds to squares, the maximum value two corresponds to diamonds, and the default value one corresponds to circles.

Period The period of the pattern, given in pixels but also displayed in physical units. All other lateral dimensions are given as fractions of this base width.

Since the patern is radial, the period must be understood as observed when travelling from the centre outwards.

Position variance Randomisation of the widths of individual ridge-trough pairs.

Parabolicity Positive values means that the terrace become narrower with increasing distance from centre and the overall profile becomes close to parabolical. Negative values have the opposite effect, with the overall shape becoming a parabola lying on its side.

Scale features with width If enabled then all lateral dimensions scale with the width of each individual ridge-trough pair for randomised patterns. Otherwise *Top fraction* or *Slope width* refer to the unperturbed base period.

Top fraction Fraction of width taken by the upper surface.

Slope width Fraction of width taken by the sloping edges. Slopes are added to the top surface; they do not replace it. Therefore, zero *Top fraction* and non-zero slope results in a triangular profile.

Asymmetry Distribution of slope width to left and right edges. Minus one means inner edges are sloped and outer edges are sharp. Plus one means inner edges are sharp and outer edges are sloped. Zero means symmetrical slopes.

Siemens star

Siemens star has the following specific parameters:

Number of sectors The number of circular sectors the plane is divided to. Each sector contain an upper and a lower surface, so the number of edges is twice the number of sectors.

Top fraction Fraction of angles taken by the upper surface.

Edge shift Shift of edges with respect to spokes meeting in the middle. Positive values mean the upper surface is enlarged by the shift, negative values mean the lower surface is enlarged. The shift does not change the asymptotic top fraction (for radius going to infinity), but it modifies the centre of the pattern.

Holes (rectangular)

Rectangular holes have the following specific parameters:

X period **and** ***Y period*** The horizontal and vertical period of the pattern, given in pixels but also displayed in physical units. All other lateral dimensions are given as fractions of the smaller of these two sizes.

Position variance Randomisation of the positions of the hole with respect to the rectangle centre.

X top fraction* and *Y top fraction Fraction of width taken by the upper surface in horizontal and vertical profiles going through the hole centre. Since the dimensions are given as two fractions, the aspect ratio of the hole scales with the corresponding two periods.

The top surface has generally larger area because some profiles do not cross any holes – and it can be further increased by round corners.

Slope width Fraction of size (smaller side of the base rectangle) taken by the sloping edges. The slope is the same in both directions.

Slopes are added to the top surface; they do not replace it. Therefore, zero top fraction and non-zero slope can result in triangular profiles.

Roundness Radius of rounded corners, given as a fraction of size (smaller side of the base rectangle).

Pillars

Pillars have the following specific parameters:

Shape The shape of the pillar base, which can be circular, square or hexagonal.

X period* and *Y period The horizontal and vertical period of the pattern, given in pixels but also displayed in physical units. All other lateral dimensions are given as fractions of the smaller of these two sizes.

Position variance Randomisation of the positions of the pillar with respect to the rectangle centre.

Size Pillar width, given as fraction the size (smaller side of the base rectangle). Pillars are symmetrical and their aspect ratio does not scale with the rectangle.

Slope width Fraction of size (smaller side of the base rectangle) taken by the sloping edges. The slope is the same in both directions.

Slopes are added to the top surface; they do not replace it. Therefore, zero top fraction and non-zero slope can result in triangular profiles.

Orientation Rotation of individual pillars with respect to the rectangular base lattice. For instance square pillars in the base orientation will look similar to the holes pattern (just inverted), whereas rotated by 45 degrees they can form a checkerboard pattern.

Double staircase

Double staircase has the following specific parameters:

Terrace X width* and *Terrace Y width The width and height of one step in the horizontal and vertical directions, given in pixels but also displayed in physical units. It can be seen as a "periods" of the pattern.

Columnar films 

The columnar film growth simulation utilises a simple Monte Carlo deposition algorithm in which small particles are incident on the surface from directions generated according to given paramerters, and they stick to the surface around the point where they hit it, increasing the height there locally. The shadowing effect then causes more particles to stick to higher parts of the surface and less particles to the lower parts. This positive local height feedback leads to the formation of columns. The algorithm considers the horizontal plane to be filled with identical copies of the surface, hence, the generated surfaces are also periodic (i.e. perfectly tilable). The surface generator has the following parameters:

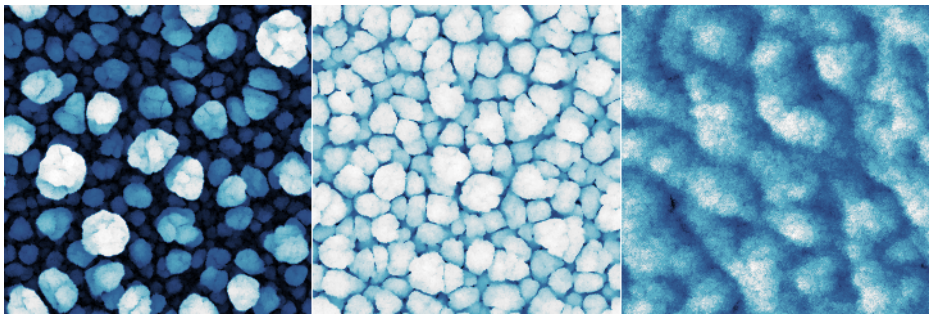
Coverage The average number of times a particle is generated above each surface pixel.

Height Local height increase occurring when the particle sticks to a pixel. Since the horizontal size of the particle is always one pixel the height is measured in pixels. A height of one pixel essentially means cubical particles, as far as growth is concerned. From the point of view of collision detection the particles are considered infinitely small.

Inclination Central inclination angle with which the particles are generated (angle of incidence). The value of zero means very small angles of incidence have the highest probability. Large values mean that the particles are more likely to impinge at large angles than at small angles. But for large direction variance, the distribution is still isotropic in the horizontal plane.

Direction Central direction in the horizontal plane with which the particles are generated. Large variance means the distribution is isotropic in the horizontal plane; for small variances the growth is anisotopic.

Relaxation Method for determination of the pixel the particle will finally stick to. Two options exist at this moment. Weak relaxation, in which only the two pixels just immediately before collision and after collision are considered and the particle sticks to the lower of them. In the case of strong relaxation, a 3×3 neighbourhood is considered in addition. The particle can move to a lower neighbour pixel with a certain probability before sticking permanently.



Artificial columnar surfaces: loosely packed columns (left); tightly packed columns (centre); directional growth with strong relaxation (right).

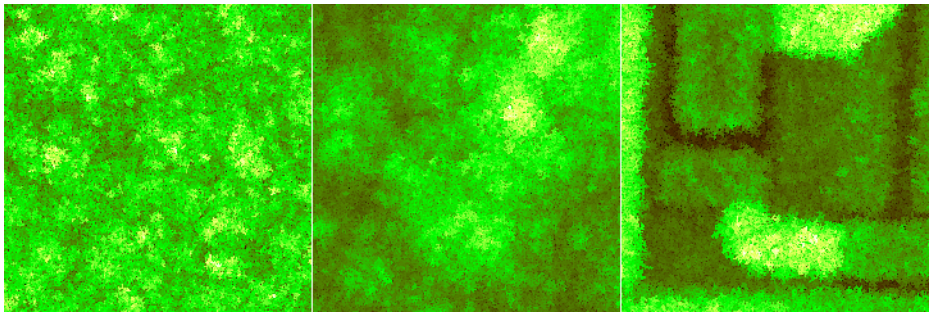
Ballistic deposition 

Vertical ballistic deposition is one of the simplest fundamental film growth models. Particles fall vertically onto randomly chosen sites (pixels). The height in the site is incremented by the particle height. However, if the new height would be smaller than the height in any of the four neigbour sites, the particle is assumed to stick to the neigbour columns. Thus height at the impact site then becomes the maximum of the neigbour heights instead. This is the only mechanism introducing lateral correlations in the resulting roughness.

The simulation has only a few parameters:

Coverage The average number of times a particle is generated above each surface pixel.

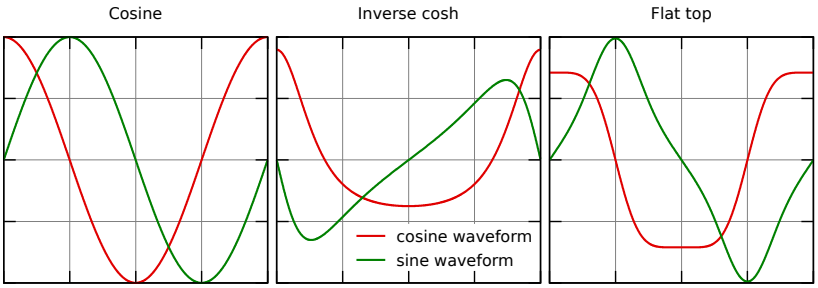
Height Local height increase occurring when the particle falls onto a pixel.



Artificial surfaces created by ballistic deposition: initial stage with relatively small correlation length (left); after correlations reached image size (centre); deposition starting from a microchip surface (right).

Waves 

The wave synthesis method composes the image using interference of waves from a number of point sources. Beside the normal cosine wave, a few other wave types are available, each having a cosine and sine form that differ by phase shift $\pi/2$ in all frequency components. When the wave is treated as complex, the cosine form is its real part and its sine form is its imaginary part.



Sine and cosine wave forms for the available wave types. Only the cosine form is used for displacement images; the entire complex wave is used for intensity and phase images.

The generator has the following options:

Quantity Quantity to display in the image. Displacement is the sum of the real parts. Amplitude is the absolute value of the complex wave. Phase is the phase angle of the complex wave.

Number of waves The number of point sources the waves propagate from.

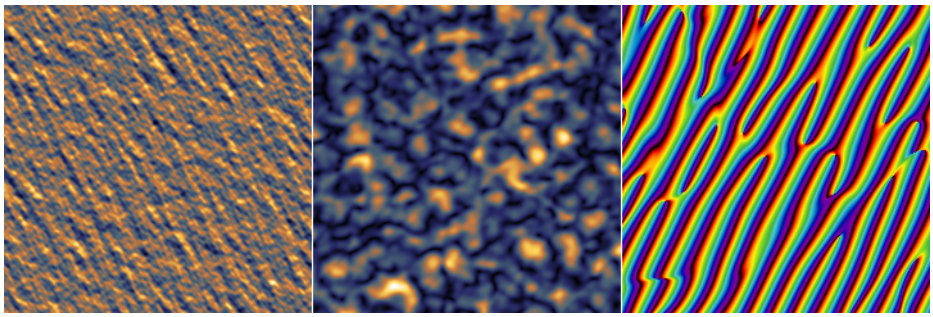
Wave form One of the wave forms described above.

Amplitude Approximate amplitude (rms) of heights in the generated image. Note it differs from the amplitude of individual waves: the amplitude of heights in the generated image would grow with the square root of the number of waves in such case, whereas it in fact stays approxiately constant unless the amplitude changes.

Frequency Spatial frequency of the waves. It is relative to the image size, i.e. the value of 1.0 means wavelength equal to the lenght of the image side.

X center, Y center Locations of the point sources. Zero corresponds to the image centre. The positions are also measured in image sizes. Generally, at least one of the corresponding variances should be non-zero; otherwise all the point sources coincide (some interesting patterns can be still generated with varying frequency though).

Decay Decay determines how quickly the waves are attenuated. The value is measured in inverse wavelengths and given as a decimal logarithm, indicated by the units log_{10}.



Artificial wave surfaces: displacement image generated with fairly large frequency variation (left); amplitude image for inverse cosh wave form (centre); phase image for a small cluster of far-away sources (right).

Domains 

The simulation implements a hybrid non-equilibrium Ising model [2], combining a discrete short-scale discrete Ising model with continuous slow inhibitor.

The discrete variable u has two possible values, between which it can flip probabilistically with probability

$$p = \min\left\{\frac{1}{2}\exp(-\Delta E/T), 1\right\}$$

where ΔE is the change of energy resulting from flipping and T is the temperature. The energy is given by the number of neighbours in the opposite state in the directions along the coordinate axes n_o, the number of neighbours in the opposite state in

the directions along diagonals n_d and also by bias caused by the inhibitor field v:

$$E = n_\mathrm{o} + \frac{1}{2} n_\mathrm{d} + Buv$$

The continuous inhibitor field v is governed by a local differential equation coupled to the variable u:

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -v - \nu + \mu u$$

where μ is the inhibitor coupling and ν is the bias parameter. The boundary conditions of the simulation are periodic, hence, the generated images are also periodic (i.e. perfectly tilable).

The calculation has the following parameters:

Number of iterations One iteration consists of four steps: A Monte Carlo update of u, a time step of solution of the differential equation for v, another Monte Carlo step and another differential euqation time step. The values of v shown correspond to the second update. The quantity shown as u is the average from the two values surrounding in time the corresponding value of v. Hence the images of u are three-valued, not two-valued.

Temperature T Temperature determines the probability with which the two-state variable u can flip to the other value if it results in a configurations with higher energy or similar (flips that lower the energy a lot occur unconditonally). A larger temperature means less separation between the two u domains.

Inhibitor strength B Strength with which the continuous variable biases the energy of each pixel. For large values the inhibitor has larger influence compared to the surface tension.

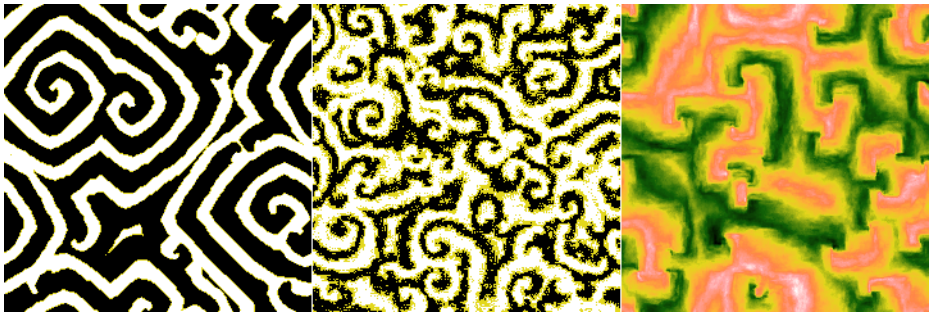
Inhibitor coupling μ Coupling factor between u and v in the differential equation for v.

Bias ν Bias in the differential equation for v towards larger or smaller values.

Monte Carlo time step Time step in the differential equation corresponding to one Monte Carlo step. So this parameter determines the relative speed of the two processes.

Height Range of values of the created images.

Tab *Presets* contains a few loadable sets of parameters producing interesting patterns. Select a preset and press *Load Selected Preset* to load the parameters to the generator. The random seed is not a part of the preset. So the output images vary with the seed, but they keep the same general character (luck is also involved; some random seeds result in less interesting images than others).



Artificial domain surfaces: spiral waves with long-range order (left); short-range ordered spiral waves at a higher temperature (centre); the continous variable for a low-temperature situation with small strength and coupling (right).

Annealing 

The simulation implements a simple cellular-automaton model of phase separation by annealing an imiscible lattice gas consisting of two components A and C. Only nearest neighbour interaction is considered. The difference of formation energies

$$\Delta E(\mathrm{AC}) = 2E_\mathrm{f}(\mathrm{AC}) - E_\mathrm{f}(\mathrm{AA}) - E_\mathrm{f}(\mathrm{CC}) \geq 0$$

determines the preference for phase separation. The annealing proceeds by two neighbour cells (pixels) swaping positions. The always relax to an energentically preferred configuration but they can also switch to energentically less preferred configuration with a probability depending on the temperature T

$$\exp\left(-\frac{\Delta E}{k_\mathrm{B} T}\right)$$

Since the probability depends only the ratio, we can put the Boltzmann equal to unity and then model depends only on a unitless temperature and relative fractions of the two components.

The calculation has the following parameters:

Number of iterations One iteration correspond to each pair of neighbour cell positions being checked for possible swapping once. However, this only holds in probabilistic sense. Some positions can be checked multiple times in one iterations, others not at all.

Initial temperature The unitless temperature determines the probability with which cells can swap into less energetically preferred configuration, as described above. The initial temperature is the temperature at which the annealing begins.

Final temperature The unitless temperature at which the simulation finishes.

Component fraction Fraction of component C in the A–C mixture.

Height Range of values of the created images.

Average iterations The output image can capture just the final state, or it can be calculated by averaging a certain number of states before the end given by this parameter.

Optionally, a three-component model can be enabled. The third component B behaves identically to A and C. However, the model now has more parameters:

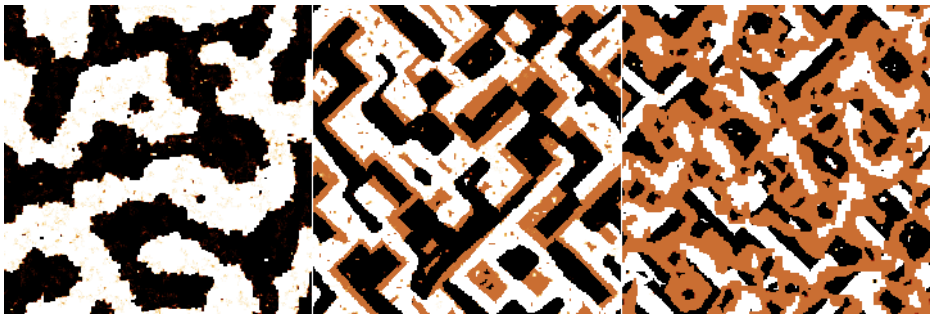
Fraction of B The fraction of component B in the mixture. The remainder is divided among A and C according to *Component fraction*.

Mixing energy AB Difference of formation energies for components A and B, relative to the maximum value.

Mixing energy AC Difference of formation energies for components A and C, relative to the maximum value.

Mixing energy BC Difference of formation energies for components B and C, relative to the maximum value.

In order to keep the unitless temperature scaled consistently with the two-component model, the largest mixing energy is always normalised to 1. If you try to decrease it, the other two mixing energies increase instead, preserving the ratio, until one of them reaches the value 1. Then this energy difference becomes the largest and the previously largest one can be decreased further.



Artificial annealing images generates for the two-component model and two different settings of the three-component model.

Coupled PDEs 

A number of interesting patterns can be generated by solving numerically coupled non-linear partial differential equations in the image plane. At present the module implements two generators with a common basic parameters:

Number of iterations The number of time steps when solving the coupled PDEs. Usually the state more or less ceases to evolve after a certain number of iterations – which, however, varies wildly with the pattern type and its parameters.

Turing pattern

The Turing pattern generator can generate the single classic pattern in a relatively controllable manner. It has the following parameters:

Size Characteristic width of stripes in the pattern.

Degree of chaos An ad hoc parameter controlling the convergence of the solution. Small values produce orderly patterns. However, the equations can take extremely long time to converge for larger sizes. Large values produce more disturbed patterns, but much more quickly.

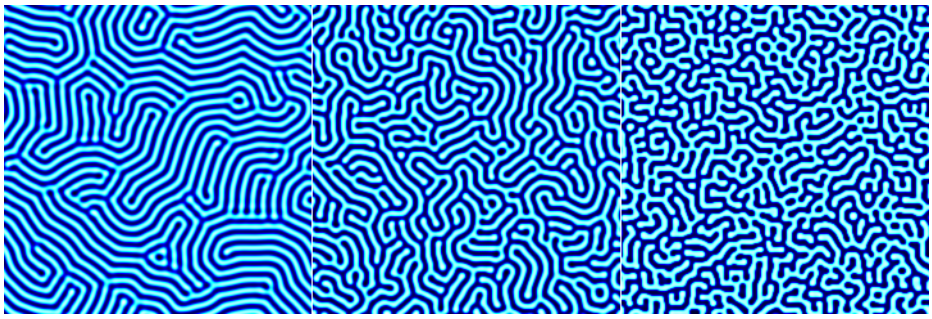
The pattern is generated by a two-component diffusion-reaction model simulation, with equations for both components of the form

$$\frac{\partial c_1}{\partial t} = p\Delta c_1 + qc_2 + rf(c_1)$$

In the equation for the second component the roles are just swapped (and the constants differ). Function f is defined

$$f(x) = x\left[\frac{100}{100+x^2} - \frac{1}{100}\right]$$

Constants p, q and r are not controlled directly. They (and the time step) are calculated from size and degree of chaos according to a calibration formula.



Three Turing patterns of the same characteristic stripe size and degree of chaos increasing from left to right.

Diffusion reaction

The diffusion reaction model can generate a variety of patterns. However, one adjusts directly the reaction constants and parameters such as feature size are not possible to control. It has the following parameters:

Output type Which of the two components should be rendered to the image.

Removal rate Removal rate constant for the reaction.

Feed rate Feed rate constant for the reaction.

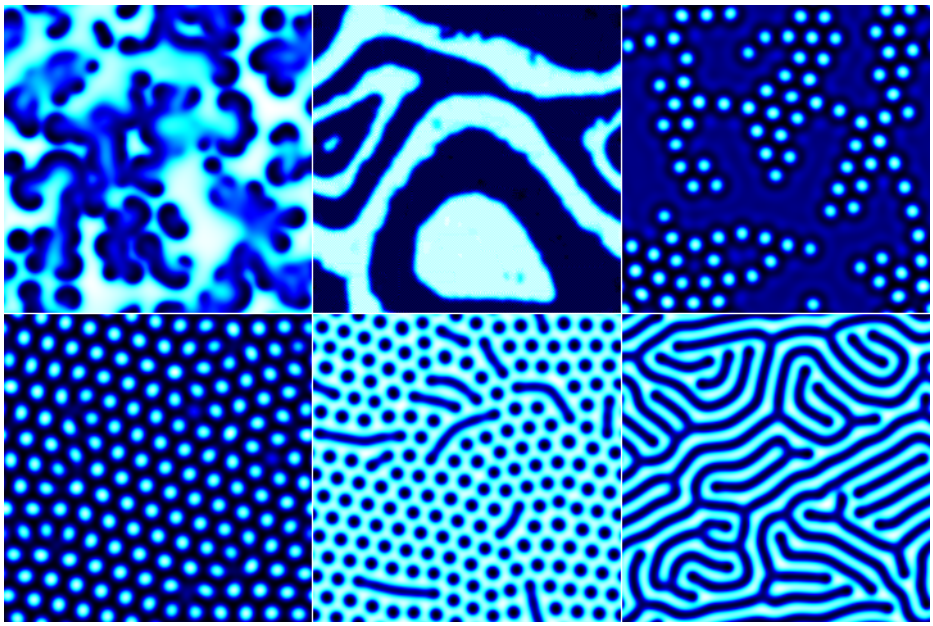
Source density The pixel density of fixed sources with low/high values. The density is per pixel, but not the factor 10^{-3}.

No-source iterations How many iterations at the end should be carried out with sources removed. It is possible to choose a number larger than the total number of iterations, although it is not very useful (the sources are removed immediately then).

In most parameter ranges the reaction needs a source of reactants to produce interesting patterns. Without such supply the system state evolves to a constant flat image or checkerboard instability. One possibility of forcing an intersting pattern (often used in demonstrations) is a large spatial variation of the reaction constants through the domain. However, this does not allow computation of specific patterns for particular parameter values.

So, instead, the module adds explicit sources. They are realised in a crude manner as random pixels which keep their low/high values (and can be sometimes observed in the animated evolution). They are not subject to the reaction, but they influence their neigbours.

Some patterns are stable once they develop and nothing happens when sources are removed. Some are not and degenerate into one of the non-interesting states. Slightly different sets of patterns can be obtained with sources removed for a long time and with sources removed just briefly at the end. In addition, there are some interesting transitional patterns.



Examples of patterns which can be produced by the diffusion reaction model.

Diffusion 

The simulation implements a Monte Carlo discrete diffusion limited aggregation model. Pixel-shaped particles fall onto the surface and move around it until they stick to a cluster, slowly forming a monolayer. The flux of incoming particles is low so that only a few particles are free to move at any given time and the particles can travel a considerable distance on the surface before sticking. For typical parameter values, this leads to the formation of "fractal flake" clusters.

The calculation has the following parameters:

Coverage The average number of times a particle falls onto any given pixel. Coverage value of 1 means the surface would be covered by a monolayer, assuming the particles cover it uniformly.

Flux The average number particles falling onto any given pixel per simulation step. Smaller values of flux lead to larger structures as the particles diffuse for longer time before they meet another particle. The value is given as a decimal logarithm, indicated by the units log_{10}.

Height The height of one particle which gives the height of the steps in the resulting image.

Sticking probability The probability that a free particle will finally stick and stop moving when it touches another single particle. Probabilities that the particle will stick if it touches two or three particles increase progressively depending on this value. The sticking probability is always zero for a particle with no neighbours and always one for a particle with all four neighbours.

Activation probability The probability a particle that has not stuck will move if it touches another particle. The probability for more touching particles decreases as a power of the single-particle probability. Particles without any neighbours can always move freely.

Passing Schwoebel probability A particle that has fallen on the top of an already formed cluster can have reduced probability to descend to the lower layer due to so called Schwoebel barrier. If this parameter is 1 there is no such barrier, i.e. particles can descend freely. Conversely, probability 0 means particles can never descend to the lower layer. The value is given as a decimal logarithm, indicated by the units log_{10}.

Note that some parameter combinations, namely very small flux combined with very small Schwoebel barrier passing probability, can lead to excessively long simulation times.

Artificial diffusion surfaces: sparse clusters with high sticking probability (left); dense clusters with low sticking probability (centre); mutltilayer fractal structures with high Schwoebel barrier (right).

Fibres 

The random fibre placement synthesis method is very similar to Objects, except instead of finite-sized objects it adds infinitely long (in principle) fibres to the surface.

The generator has quite a few parameters, divided into *Generator* and *Placement*. The first group contains the basic settings and parameters determining the profiles of individual fibres:

Shape The shape of profile across a fibre. Options *Semi-circle*, *Triangle*, *Rectangle* and *Parabola* are self-explanatory. *Quadratic spline* shape corresponds to the three-part piecewise quadratic function that is the basis function of quadratic B-spline.

Coverage The average number of times a fibre would be placed onto any given pixel. Coverage value of 1 means the surface would be exactly once covered by the fibres assuming that they covered it uniformly and did not overlap. Values larger than 1 mean more layers of fibres – and slower image generation. The value is approximate and the actual coverage depends somewhat on other parameters.

Width Fibre width in pixels. It has the usual *Variance* associated, which determines the difference between entire fibres. However, the width can also vary along a single fibre. This is controlled by the *Along fibre* option. High values of variation along the fibre can in combination with some other options (in particular high lengthwise deformation and height variation) result in very odd looking images and possibly artefacts.

Height A quantity proportional to the height of the fibre, normally the height of the highest point. Similarly to *Width*, it has two variances, between entire fibres and along each individual fibre.

Enabling *Scales with width* makes unperturbed heights to scale proportionally with fibre width. Otherwise the height is independent on size.

Button *Like Current Image* sets the height value to a value based on the RMS of the current image.

Truncate The fibres can be truncated at a certain height, enabling creation of profiles in the form of truncated semi-circles, parabolas, etc. The truncation height is given as a proportion to the total fibre height. Unity means the fibre is not truncated, zero would mean complete removal of the fibre.

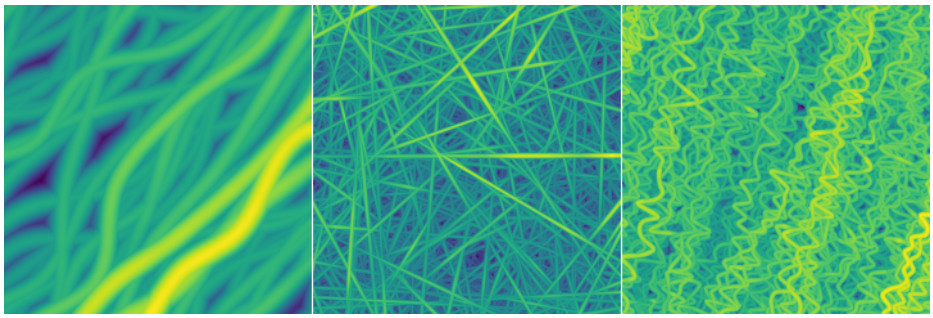
The second group contains options controlling how fibres are deformed and added to the image:

Orientation The rotation of fibres, measured anticlockwise. Zero corresponds to the left-to-right orientation.

Density Density controls how frequently the fibre is deformed along its length. Low density means gradual deformation on a large scale, whereas high density means the fibre is deformed often, leading to wavy or curly shapes.

Lateral How much the fibre is deformed in the direction perpendicular to its main direction. Increasing this value makes the fibre wavy, but in a relatively regular manner.

Lengthwise How much points of the fibre are moved along its length. Increasing this value causes irregular kinks and loops (or even knots) to appear.



Artificial fibre surfaces: moderately deformed fibres, all oriented more or less in the same direction (left); many layers of thin straight randomly oriented fibres (centre); parallel fibres with high lateral deformation density (right).

Lattice 

The lattice synthesis module creates surfaces based on randomised two-dimensional lattices. It has two major parts. The first part, controlled by parameters in tab *Lattice*, is the creation of a set of points in plane that are organised to a more or less randomised lattice. The second part, controlled by parameters in tab *Surface*, is the actual construction of a surface based on quantities calculated from Voronoi tessellation and/or Delaunay triangulation of the set of points.

The creation of the lattice has the following parameters:

Lattice The base lattice type. The random lattice corresponds to completely randomly placed points. Other types correspond to regular organisations of points (square, hexagonal, triangular), well-known tilings (Cairo, snub square, Penrose) or surface reconstructions (silicon 7×7).

Size The average cell size. More precisely, this parameter describes the mean point density. It is equal to the side of the square if the same number of points was organised to a square lattice.

Lattice relaxation Amount to which the lattice is allowed to relax. The relaxation process pushes points away from very close neighbours and towards large empty space. The net result is that cell sizes become more uniform. Of course, relaxation has no effect on regular lattices. It should be noted that relaxation requires progressive retessellation and large relaxation parameter values can slow down the surface generation considerably.

Height relaxation Amount to which the random values assigned to each point (see below) are allowed to relax. The relaxation process is similar to diffusion and leads to overal smoothing of the random values.

Orientation The rotation of the lattice with respect to some base orientation, measured anticlockwise. It is only available for regular lattices as the random lattice is isotropic.

Amplitude, ***Lateral scale*** Parameters that control the lattice deformation. They have the same meaning as in Pattern synthesis.

The final surface is constructed as a weighted sum of a subset of basic quantities derived from the Voronoi tessellation or Delaunay triangulation. Each quantity can be enabled and disabled. When it is selected in the list, its weight in the sum and its thresholding parameters can be modified using the sliders. The lower and upper threshold cut the value range (which is always normalised to $[0, 1]$) by changing all values larger than the upper threshold equal to the thresold value and similarly for the lower thresold.

Some of the basic quantities are calculated from lateral coordinates only. Some, however, are calculated from random values ("heights") assigned to each point in the set. The available quantities include:

Random constant Rounding interpolation between the random values assigned to each point of the set. This means each Voronoi cell is filled with a constant random value.

Random linear Linear interpolation between the random values assigned to each point of the set. Thus the surface is continuous, with each Delaunay triangle corresponding to a facet.

Random bumpy An interpolation similar to the previous one, but non-linear, creating relatively level areas around each point of the set.

Radial distance Distance to the closest point of the set.

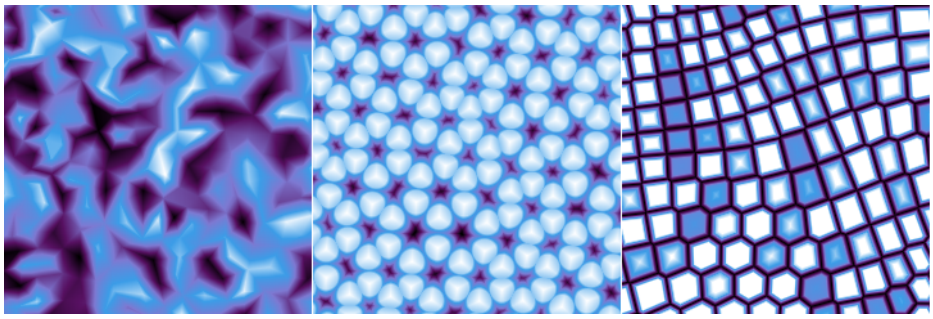
Segmented distance Distance to the closest Voronoi cell border, scaled in each cell segment so that the set point is in the same distance from all borders.

Segmented random The same quantity as segmented distance, but multiplied by the radndom value assigned to the point of the set.

Border distance Distance to the closest Voronoi cell border.

Border random The same quantity as border distance, but multiplied by the radndom value assigned to the point of the set.

Second nearest distance Distance to the second nearest point of the set.



Artificial lattice surfaces: faceted surface created by random linear interpolation (left); bumps in a distorted triangular pattern (centre); distorted rectangular pattern with segments separated by trenches (right).

Brownian 

The module generates, among other things, surfaces with profiles similar to samples of fractional Brownian motion. The construction method is, however, not very sophisticated. Starting from corners of the image, interior points are recursively linearly interpolated along the horizontal and vertical axes, adding noise that scales with distance according to the given Hurst exponent. Some of the surfaces are essentially the same as those generated using spectral synthesis, however, constructing them in the direct space instead of frequency space allows different modifications of their properties.

The generator has the following parameters:

Hurst exponent The design Hurst exponent H. For the normal values between 0 and 1, the square root of height-height correlation function grows as H-th power of the distance. The construction algorithm permits formally even negative values beause it stops at the finite resolution of one pixel.

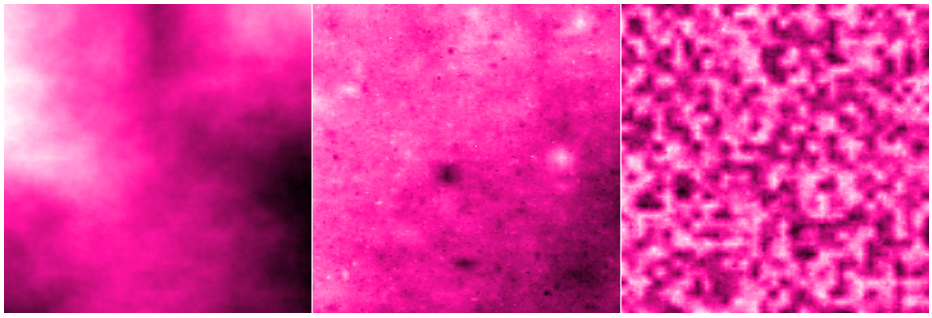
Stationarity scale Scale at which stationarity is enforced (note Fractional Brownian motion is not a stationary). When this scale is comparable to the image size or larger, it has little effect. However, when it is small the image becomes "homogeneised" instead of self-similar above this scale.

Distribution Distribution of the noise added during the generation. Uniform and Gaussian result in essentially the same surfaces (statistically); the former is faster though. More heavy-tailed distributions, i.e. exponential and especially power, lead to prominent peaks and valleys.

Power Power α for the power distribution. The probability density function is proportional to

$$\frac{\alpha/2}{(|z|+1)^{\alpha+1}}$$

RMS The root mean square value of the heights (or of the differences from the mean plane which, however, always is the $z=0$ plane). Note this value applies to the specific generated image, not the process as such, which does not have finite RMS. Button *Like Current Image* sets the RMS value to that of the current image.



Artificial Brownian surfaces: classic surface generated with H about 0.7 (left); nebula-like image generated with a heavy-tailed power distribution (centre); the effect of small stationarity scale (right).

Phases 

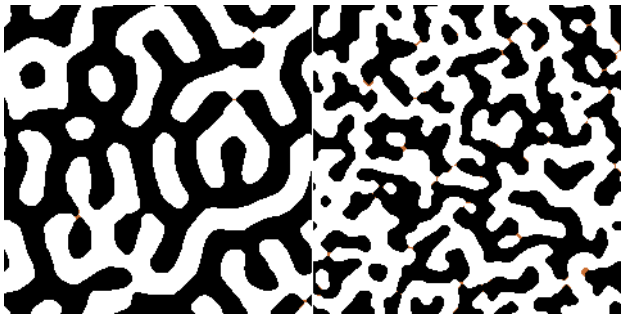
The module creates data somewhat resembling magnetic domain structures that have been studied by magnetic force microscopy (MFM). Unlike more physically correct but very computationally expensive approaches, it works by spectral synthesis of a surface with a limited range of spatial frequencies, and then application of morphological operations to obtain a two-phase image. The output image thus mostly contains of two values representing the two phases, although occasionally a middle "in transition" value can appear.

The generator has the following parameters:

Size Typical width of the stripe.

Size spread Spread of sizes (or, more precisely, spatial frequencies). Small spread means smooth regular features. However, for very small spread values preferred directions may appear in the image (it becomes anisotropic) due to the exhaustion of allowed spatial frequencies. Large spread means less regular phase boundaries and also more in-transition regions.

Height The difference between the values corresponding to the two phases, determining overall z-scale of the image.



Artificial two-phase images: small size spread (left); larger size spread (right).

Discs 

The module creates data consisting of non-overlapping discs, possibly post-processed to form a surface covered by tiles. A similar result can be also achieved with Objects when *Avoid stacking* is enabled. However, in this case the only a few seed discs are placed randomly, the remaining ones are always placed to the middle of the largest remaining empty space. This results in a more uniform distribution of sizes. Furthermore, the transformation to tiles creates structures where typically three tiles meet at angles close to 120 degrees, similarly to foam structures.

The generator has the following parameters:

Starting radius Radius of the seed discs placed randomly at the beginnig.

Minimum radius The minimum allowed radius of all other discs. When there is no space for a disc of at least this size, the generation stop.

Separation Additional separation from neighbour discs. While *Minimum radius* detemines the minimum size of the actually placed dics, this parameter increases the minimum size of free space allowing placement.

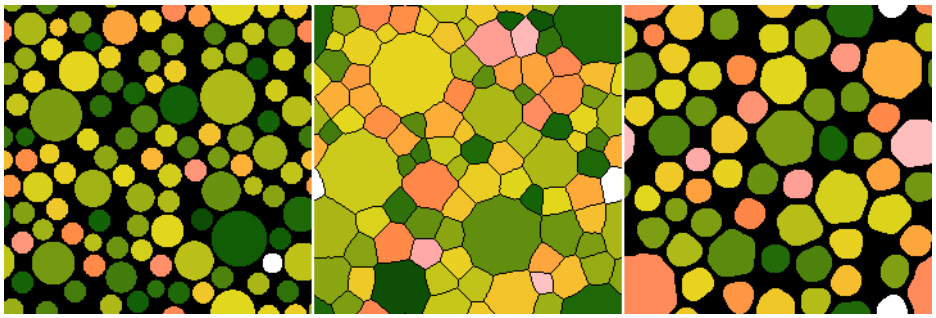
Transform to tiles When enabled, all discs are expanded until they would touch their neighbours. The surface becomes tiles, with thin gaps between tiles.

Gap thickness Width of gaps between tiles after the tile transformation.

Apply opening filter When enabled, the tiling is further post-processed using opening filter of given size. This removes too small tiles and makes the remaning ones rounder.

Size Size of the opening filter.

Height The height of discs or tiles with respect to the base, determining overall z-scale of the image. Heights of individual features will be spread around this value when *Variance* is non-zero.



Artificial disc images: round discs with some height distribution (left); transformation to tiles (centre); erosion by opening filter (right).

Dunes 

The module is based on a simple simulation of formation of eolian dunes [3]. Particles can be picked up by wind with a certain probability, transferred along the wind direction and deposited in another location. The function has the following parameters:

Coverage The average number of particles per one pixel.

Number of iterations One iteration correspond to wind picking up randomly one particle in each pixel and deposting it elsewhere. However, the pixels are chosen randomly. Some locations may be chosen multiple times in one iteraton, others not at all. Some selected locations may also be bare bedrock with no sand particle to pick up.

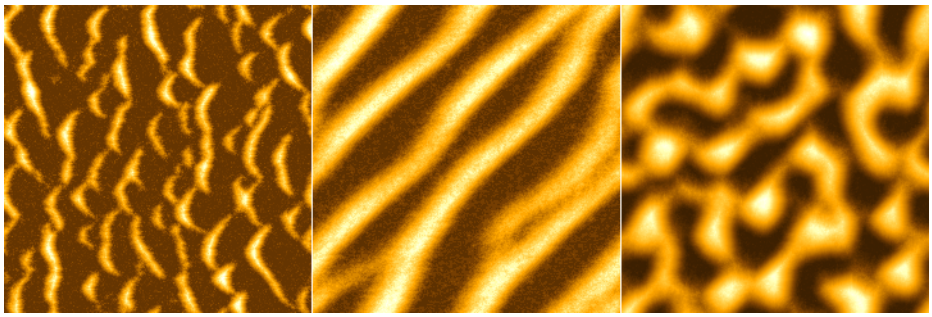
Direction* and *Spread The mean wind direction and wind direction spread. Small spread means the wind direction is consistent. Large values mean more random wind direction.

Minimum step* and *Step range The sand particle can be imagined as only touching the surface occasionally and being carried by the wind high above it between. It can only be deposited when it touches the surface. The minimum step is the minimum distance (in pixels) of one such jump; the range gives the interval (again in pixels) from minimum to maximum jump length.

Stickng probability on rock The probability the particle is deposited when it touches the bare bedrock.

Stickng probability on sand The probability the particle is deposited when it touches the surface and there is already some sand. This probability should be generally higher than for bare rock.

Maximum slope When deposition of a particle (or picking up by wind) creates a too large height difference between neighbour pixels the sand becomes unstable. It then trickles down to either reduce the slopes of a hill for fill holes until the maximum slope is nowhere exceeded.



Dune images: the default settings (left); long flat dunes with small maximum slope (centre); more isotropic dunes for highly random wind direction (right).

Plateaus 

The surface is created by stacking flakes or plateaus on top of each other, starting from the largest and progressing to the smallest according to a scaling law. By default they do not overlap, but it can be enabled to allow overlapping flakes (a bit like those that might occur with 1D materials such as graphene). The generator has the following parameters:

Maximum size The approximate maximum radius of a plateau.

Minimum size The approximate minimum radius of a plateau.

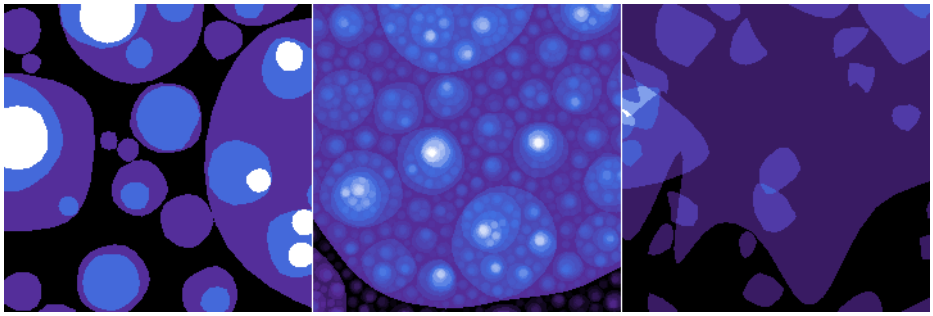
Size power law How fast the plateau size radius decreases with the increasing number of plateaus already there. High values mean the size decreases quickly and plateaus tend to be sparse and isolated. Low values mean the size decreases slowly and plateaus ten to be dense and stacked high.

Warning It may be difficult, or even impossible, to find positions for all the plateaus with given the maximum and minimum size and scaling power. Low powers are only useful with narrow size ranges. Otherwise the surface generation can be extremely slow – or even never finish at all.

Shape irregularity How much the shapes can differ from symmetrical discs.

Overlap fraction Approximate fraction of plateaus that overlap with another plateau.

Scale with power of size The power with which plateau height scales with its radius. For the default power of 0 all pleateaus have the same height (apart from possible random spread). Positive powers mean the height decreases as the size decreased. Negative powers mean smaller plateaus are taller, creating a spiky surface.



Plateau images: the default settings (left); approximately self-affine surface with small scaling power and minimum size (centre); somewhat overlapping irregular flakes (right).

Residue

The function generates structures resembling resiude of a partially disolved thin film on a flat surface. It is achieved by simple removal of circular regions, either connected or disconnected from already removed parts. It has a the following parameters:

Coverage The total area to remove. It is a slight misnomer, because small values mean only a small fraction of the film is dissolved – therefore a lot of the surface is still covered by the residue. It is, nevertheless, consistent with how the term is used in other modules.

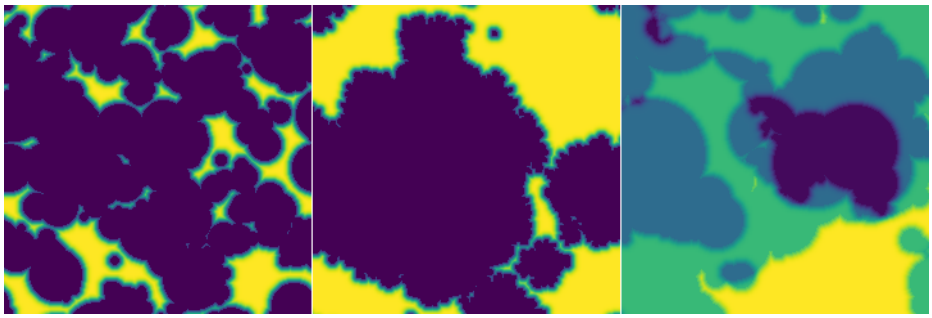
Probability of a new location The probability that a new independent location is chosen for disolving the film (it may still connect to existing holes by chance). Low values give connected lichen-like holes. Large values give random holes.

Minimum radius The minimum radius of circular areas to dissolve.

Maximum radius The maximum radius of circular areas to dissolve.

Size power law Power with which radius probability decreases. Small values produce mostly only large discs; large values produce mostly only small ones. The default value of 2 corresponds to approximately equal total areas of discs of all sizes.

Edge width Width of hole edges in pixesl. Zero leads to a bi-level image. Large values means disolution continues far outside the holes and the outside resembles Euclidean distance to the nearest hole.



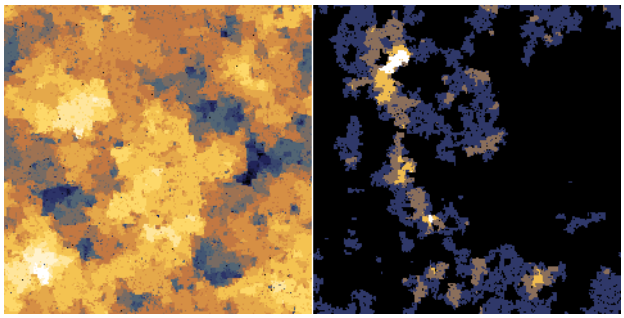
Film residue images: relatively large coverage with otherwise default settings (left); connected holes formed by discs with mostly small radii (centre); multilayer image created by repeated execution (right).

Wetting

Wetting is a simple simulation of wetting front progressing through a random medium. The medium is represented as a voxel grid, where each voxel has a different resistance. The liquid always propagates to the least resistant voxel. The generator produces isotropic patterns with discrete heights corresponding to the levels to which the wetting front have progressed. It has a couple of parameters:

Coverage The average number of times the liquid is propagated per pixel. The resulting image shows only the farthest points, so its average value is not in a simple relation to coverage.

Diffusion The probability of random propagation from neighbour voxels, not according to the resistance based priority. Note that the value is logarithmic.



Wetting front images: a typical result with moderate coverage (left); initial stage of propagation (right).

References

[1] D. Nečas, P. Klapetek: Synthetic Data in Quantitative Scanning Probe Microscopy. Nanomaterials 11 (2021) 1746 10.3390/-nano11071746

[2] L. M. Pismen, M. I. Monine, G. V. Tchernikov: Patterns and localized structures in a hybrid non-equilibrium Ising model. Physica D 199 (2004) 82, doi:10.1016/j.physd.2004.08.006

[3] B. T. Werner: Eolian dunes: Computer simulations and attractor interpretation. Geology 23 (1995) 1107, doi:10.1130/0091-7613(1995)023<1107:EDCSAA>2.3.CO;2

4.30 Calibration and uncertainties

Calibration data

Calibration data can be used to provide correction of measured data or perform uncertainty calculations. Generally, calibration data can be of different types and different levels of complexity. For most of the cases user acquires error in each axis, e. g. using a calibrated standard. This value can be used for data correction. Similarly, the value of uncertainty is mostly determined for each axis from calibrated standard certificate and from measurement process uncertainty budget.

In more complex cases, calibration data can be determined locally. Scanner error cannot always be described by only three parameters (one for each axis) and its uncertainty is not necessarily the same in whole range. For precise measurements it is

therefore practical to determine local errors and namely local uncertainties that can be used for further calculations. By "local" we mean here uncertainties that are related to certain location in the complete volume that can be reached by the scanner.

To obtain local errors and uncertainties, one can use a calibration standard again or use a more complex instrument, like interferometer for scanning stage calibration. This is usually done in metrology institutes.

In Gwyddion, there is a set of tools helping local uncertainty processing. Primary calibration data, related to a scanning stage, can be determined or loaded. They can be assigned to a certain SPM measurement data creating a set of calibrations. These are used automatically in tools and modules where uncertainty propagation calculation can be performed in order to provide measurement uncertainty.

Calibration data acquisition

Data Process → *Calibration* → *3D calibration*

Calibration data can be acquired in the following ways:

The module *3D calibration* → *Create...* can be used for creating simplest primary calibration data - based only on knowledge of xyz errors and uncertainties and on scanner volume geometry. We also need to specify calibration name. Primary calibration data will be accessible under this name module for its data application to SPM measurements.

Using *3D calibration* → *Get simple error map...* module, we can determine primary xyz calibration data from a set of calibration grating measurements. Here we need to provide several measurements of calibration grating for different z-levels of the scanner. This forms several cross-sections of the full scanner volume. Providing single grating detail, nominal grating pitch and assuming that the grating planarity and orthogonality is much higher than that of scanning stage we can determine primary calibration data on the basis of correlation. Note that this way of calibrating a microscope is only very approximate and its use for metrology purposes is very limited. However, it can provide a lot of useful information about our scanner properties if we are unable to perform more complex analysis.

Finally, using *3D calibration* → *Load from file...* we can load any primary 3D calibration data determined by an external device, like set of interferometers. Data should be a plain text file containing number of calibration sets and sets (x, y, z, x_err, y_err, z_err, x_unc, y_unc, z_unc).

Calibration data application and further use

Data Process → *Calibration* → *3D calibration*

Primary calibration data obtained in previous steps are related to a scanning stage, not to concrete SPM measurements. We can use primary calibration data for different measurements. To use primary calibration data for our measured data processing, we need to apply them first. Using module *3D calibration* → *Apply to data...* we can review and select calibration data applied on our height field measurements. After applying calibration data a set of calibration datafields is created and attached to selected data. A 'C' character appears in data browser as a sign of data with attached calibration. Note that the calibration attached to SPM measurement is no more related with primary calibration data (that were used for its creation).

When there is calibration attached to data, data reading and statistical quantities evaluation modules and tools recognize it automatically. Measurement uncertainty is then added to the measurement results. Uncertainty is calculated using uncertainty propagation law.

4.31 Python Scripting

Note Pygwy only works with Python 2.7. Any Python 3 version that might exist in future will have to be created from scratch and will not be backward compatible.

Python scripting allows you to automate data SPM processing tasks, both within Gwyddion and by using its functionality in standalone Python scripts run outside Gwyddion. The scripts can range from simple macro-like executions of several functions in sequence to complex programs combining data from multiple files and utilising third-party libraries or programs.

The Gwyddion module providing this functionality is called *pygwy* and the term pygwy is by extension used also for Python scripting with Gwyddion in general, even if not directly related to the pygwy module itself.

The term "Python scripting" implies Gwyddion scripts are written in Python. Python is a powerful and widely used high-level, general-purpose, interpreted programming language with concise, easy-to-understand syntax, wide set of libraries and support for a large number of operating system and environments. It is also well documented with tutorials available. And all the available Python features and facilities can be used in Gwyddion Python scripts. The only new thing is that you also gain access to Gwyddion data structures, functions and program state.

Extending and embedding, modules and modules

Since you can combine Gwyddion and Python in several different ways and both programs also use the term "module" extensively, and for different things, a clarification is in order.

Gwyddion provides a Python extension module called `gwy`. So you can run Python, load the `gwy` module the same way as any other module using

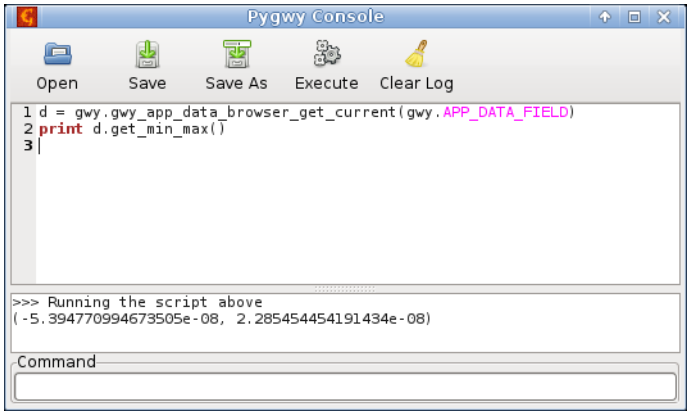
```
import gwy
```

and start utilising its functions and classes in the Python shell or a script. This is usually called a *standalone* pygwy script. Of course, you are now running Python, not Gwyddion. So you cannot use functionality related to interaction with Gwyddion (the program). Even if a Gwyddion instance is running simultaneously it is completely independent on the script and cannot be directly controlled by it.

On the other hand, the pygwy module (a Gwyddion module) allows combining things the other way round and running the Python interpreter inside Gwyddion, i.e. embedding Python in Gwyddion. When you execute *Data Process* → *Pygwy Console* a Python scripting console appears from which you can run Python commands. In this case the Python code can interact with the current Gwyddion instance in various ways. A script run in this manner is usually simply called pygwy script (without any epithet).

The *Command* entry in the lower part of the console window allows executing simple individual commands. They are immediately executed when you press **Enter** and their output is printed in the log area above.

Longer scripts can be typed, pasted or loaded into the main area in the upper part. The button *Execute* (**Ctrl-E**) then runs the script. The output is again displayed in the log area, together with any error messages. The other control buttons enable saving (**Ctrl-S**) and loading (**Ctrl-O**) the scripts. Button *Clear Log* clears the log area.



The pygwy console showing a simple script (printing the range of values of the current image) and its output.

There are some other subtle differences from standalone scripts. The `gwy` module is always automatically imported in the console. In addition, it will be available there even if you find the Python extension module on disk and delete it because here it is created on the fly from within Gwyddion.

Finally, you can write Gwyddion extension modules in Python. Such modules will appear in the *Data Process* menu, similar to other data processing functions. You can also write modules that implement loading and saving of data in new formats. This is slightly more involved than the previous two options and will be explained below. This kind of module is usually called Python data processing module (with Gwyddion context implied), or Gwyddion Python data processing module if the context is unclear.

In addition to the main `gwy` module, there is also a small module `gwyutils` that offers some extra auxiliary functions. A part of them provide workaround for functionality that used not to be covered well by `gwy` itself and are mostly obsolete. Some are, however, still useful. The `gwyutils` module is not imported automatically; you always have to import it explicitly with

```
import gwyutils
```

Depending on how Gwyddion was installed, it may be necessary to add the path to `gwyutils.py` to `sys.path` beforehand:

```
import sys
sys.path.append('/the/path/to/the/gwyutils/module')
```

Data fields – how images are represented

The probably most important Gwyddion data structure you will encounter is DataField that represents an image.

It is essentially an array of $N \times M$ floating point data values that also carries additional information, such as physical dimensions or units. The data values are always in base SI units (e.g. metres or Volts, nor nanometres or millivolts). In most cases you will work with DataFields that came from a file. However, creating a new zero filled 128 by 128 data field (with physical dimensions one by one) is not difficult:

```
data_field = gwy.DataField(128, 128, 1.0, 1.0)
```

Pixel dimensions are called resolutions in Gwyddion and can be obtained using functions `get_xres()` and `get_yres()`:

```
# Pixel width
xres = data_field.get_xres()

# Pixel height
yres = data_field.get_yres()
```

Or you can create a data field as a copy of another

```
another_field = data_field.duplicate()
```

or one that is similar to another data field but zero filled

```
similar_field = data_field.new_alike()
```

The DataField object has lots of methods, from simple, such as obtaining the physical dimensions or simple statistical quantities

```
# Physical dimensions
xreal, yreal = data_field.get_xreal(), data_field.get_yreal()

# Value range
zmin, zmax = data_field.get_min_max()

# Mean value
mean = data_field.get_avg()
```

to complex ones performing complicated operations with many tunable parameters, such as marking of scars (strokes)

```
data_field.mark_scars(result_field, high, low, min_len, max_width, False)
```

You can find them all in the reference documentation. Some common operations may involve combining a few functions together. For instance the equivalent of Fix Zero can be done by combining `get_min()` and `add()`:

```
data_field.add(-data_field.get_min())
```

If you do more complex data processing, just using the existing methods may not be sufficient and you will need to read or modify the individual pixel values directly. There are several possible methods. The simplest is direct indexing; to read the value in the top-left corner you can just use

```
value = data_field[0]
```

and similarly

```
data_field[0] = 1.5
```

sets the top-left corner value to 1.5. Note that the image data are "flattened", i.e. present in one big array, ordered row by row, and even though DataField represents an image only one index is used (as opposed to separate row and column indices). So, the last value on the third row of the 128 by 128 data field could be changed to -1.0 by

```
data_field[2*128 + 127] = -1.0
```

Another possibility is the methods `get_data()` and `set_data()` that extract the entire data to a Python list and fill the entire DataField data from a Python list (or other sequence), respectively. Note these method always copy the data. If you change the extracted list it does not change the data field's data and vice versa. For instance if you want to get your hands dirty Fix Zero can be replicated as follows:

```
data = data_field.get_data()
m = min(data)
data = [z-m for z in data]
data_field.set_data(data)
```

Finally, if you use NumPy you might want to directly operate on the data field's using NumPy functions. The helper module `gwyutils` provides functions that enable this. Calling

```
array = gwyutils.data_field_data_as_array(data_field)
```

creates a NumPy array that references the data field's data. This has to be used with care because some DataField operations change the data representation (for instance resizing) and then the NumPy array breaks. Generally, it is not recommended to mix haphazardly operations with the array and DataField methods.

When you are done modifying a data field corresponding to an image in Gwyddion you should call

```
data_field.data_changed()
```

This emits a signal of the DataField object notifying any data windows and thumbnails displaying the image that they should update themselves. Generally, this should be only done once for each changed data field at the end of the script (or when you are finished with this particular image). If you forget this the display may get out of sync with the real image content.

Containers – how files are represented

An SPM file is represented by Container which is a dictionary-like object holding everything that can be present in a GWY file. Every file is upon import transformed to a Container with the same structure. So there is no difference between GWY files and other file types once they are loaded into the program.

Items in a Container are identified by string keys (or equivalent integer keys, though in Python this is used more seldom), for instance `"/0/data"` identifies image number 0 and `"/5/select/rectangle"` identifies rectangular selection on image number 5. The locations of the various pieces of data are described in the GWY file format specification.

You can access items in a Container by simple indexing:

```
# Set image 0 to our data field
container['/0/data'] = data_field

# Get rectangular selection for image 5
selection = container['/5/select/rectangle']
```

However, Container differs from a Python dictionary in several aspects. It can hold only certain data types: booleans, integers, floating point values, strings and serialisable Gwyddion objects (which is essentially any Gwyddion data object you can come across). It has methods such as `set_object_by_name()` or `get_boolean_by_name()` that emphasise the specific type of data being stored or extracted and you can use them instead of indexing. Furthermore, even with these limitations if you start storing random things with random names there you will not obtain something Gwyddion understands as an SPM file and most likely confuse it to no end, possibly even crash. So always stick to item names given in the specification and the corresponding item types.

There are functions that help with construction of keys identifying various items in the file, called `gwy_app_get_`*something*`_key_for_` where "something" stands for the item type. So the key for image 3 can be constructed using

```
key = gwy.gwy_app_get_data_key_for_id(3)
```

Note that these functions construct the integer keys (that work equally well but are opaque). You can use functions `gwy_name_from_key` and `gwy_key_from_name()` to convert between integer and string keys. Calling

```
gwy.gwy_name_from_key(gwy.gwy_app_get_data_key_for_id(3))
```

will produce `"/3/data"` as expected.

The list of everything inside a Container can be obtained using methods `keys()` that produces a list of integer keys, or `keys_by_name()` that produces a list of string identifiers. This can be occasionally useful if you need to scan the container. In most cases, however, functions listing individual data kinds are more convenient.

Other data types

A Gwyddion file does not contain just images. It can also contain masks, selections, volume data, graphs, XYZ data, ... So here we give a brief overview of other objects you may reasonably encounter.

DataField is also used to represent masks and presentations. They must always have the same dimensions as the main data field. In mask data fields the value 1 (or larger) stands for masked and 0 (or smaller) for unmasked. Values between 0 and 1 should be avoided. In presentations the value range does not matter. They are stored under keys of the form `"/0/mask"` or `"/0/show"`.

Container also represents metadata. Metadata Containers have different rules: all the metadata values must be strings. Otherwise both keys and values are free-form.

Brick represents volume data. It is very similar to DataField except it has three dimensions except of two.

DataLine represents conversely regularly spaced one-dimensional data. You will not actually encounter it in GWY files (curve plots are represented differently) but it is used in various data processing functions.

Spectra represents set of spectra curves, i.e. single point spectroscopy data.

SIUnit represents physical unit. It is an auxiliary object that you will not encounter standalone but all kinds of data objects carry SIUnits specifying the physical units of dimensions or values.

GraphModel represents a graph containing a set of curves.

GraphCurveModel represents a single curve in a graph.

Selection represents a selection on an image or graph. It is a sequence of coordinates. Selection is in fact an abstract base class, each type of selection (point, line, rectangle, ...) is a separate subclass.

Finding and enumerating data

Knowing everything about Containers, DataFields and SIUnits, you would probably also like to know how to get the data to process.

In the context of scripts or modules that should work with the currently active data, as normal Gwyddion data processing modules do, the question is how to obtain the currently active data. The answer is function `gwy_app_data_browser_get_current()`. It takes a AppWhat argument specifying what kind of current item you are interested in and returns the appropriate item. It can be used to obtain the data objects themselves

```
# Get the active image (DataField)
data_field = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD)

# Get the active file (Container)
container = gwy.gwy_app_data_browser_get_current(gwy.APP_CONTAINER)
```

or their number in the file

```
# Get the active image number
i = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD_ID)

# Get the active graph number
i = gwy.gwy_app_data_browser_get_current(gwy.APP_GRAPH_MODEL_ID)
```

or their integer keys

```
# Get the active image key
key = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD_KEY)

# Get the active mask data field key
key = gwy.gwy_app_data_browser_get_current(gwy.APP_MASK_FIELD_KEY)
```

and also some less common things such as the active page in the data browser. When there is no active item of the requested type the function returns None.

On the other hand, frequently you need to find all data of certain kind in the file. The list of all image (channel) numbers can be obtained with

```
ids = gwy.gwy_app_data_browser_get_data_ids(container)
```

When the function returns the list `[0, 1, 4]` it means the container contains images with numbers 0, 1 and 4. In other words, there will be images under keys `"/0/data"`, `"/1/data"` and `"/4/data"`. Similar functions exist for enumeration of graphs, spectra, volume or XYZ data. There is also the function `gwy_app_data_browser_get_containers()` that produces the list of all files (i.e. Containers representing files) currently open.

Finally, it is often useful to locate data objects by title. A simple search function `gwy_app_data_browser_find_data_by_title` is provided. It matches the titles using wildcards similar to Unix shell:

```
# Find images with title exactly "Height"
ids = gwy.gwy_app_data_browser_find_data_by_title(container, 'Height')

# Find images with title starting with "Current"
ids = gwy.gwy_app_data_browser_find_data_by_title(container, 'Current*')

# Find images with title containing "03"
ids = gwy.gwy_app_data_browser_find_data_by_title(container, '*03*')
```

When you add new image or other data type to a file use a data browser function:

```
i = gwy.gwy_app_data_browser_add_data_field(container, data_field, True)
```

The last parameter specifies whether to show the data in a data window or not. The function returns the number assigned to the new data field. You can then use it to construct identifiers of related data. Similar functions exist for other data types.

Running module functions

Utilisation of DataField (and other) methods, perhaps combined with a bit of direct data access, is the most straightforward and often preferred approach to data processing in a pygwy script. Nevertheless, sometimes you also want to run a Gwyddion function exactly as if it was selected from the menu. Either because the function is not directly available as a DataField method nor easily replicated, or you prefer writing a macro-like script that executes various existing Gwyddion functions.

A data processing function (corresponding to a menu or toolbox item) is run using `gwy_process_func_run()`:

```
gwy.gwy_process_func_run('level', container, gwy.RUN_IMMEDIATE)
```

The first argument `'level'` is the function name, the second is the container (which should generally be always the current container to prevent strange things from happening) and the last argument is the mode in which to run the function. Data processing functions have two possible modes `RUN_INTERACTIVE`, which means if the function has any dialogue it should show it, and `RUN_IMMEDIATE`, which means the function should not display any dialogues and immediately perform the operation. Note that not all functions can be run in both modes. Some have no user interface, while others can only be run interactively.

How the function knows which data to process? It processes the current data. For macro-like scripts or data processing functions written in Python this is exactly what you want. You can also select (make current) another data in the script, as described in the following section.

Where will the result end up? Some functions modify existing data, some create new images (and other types of output). Some even allow choosing between these two options. Whatever the function does when run from the GUI it also does when run from a pygwy script. If new images are created you need to find them. This can be done as follows

```
ids_before = set(gwy.gwy_app_data_browser_get_data_ids(container))
gwy.gwy_process_func_run('some_func', container, gwy.RUN_IMMEDIATE)
ids_after = set(gwy.gwy_app_data_browser_get_data_ids(container))
new_ids = ids_after - ids_before
```

Usually the set `new_ids` will have single member, the numeric identifier of the newly created image, but some modules may create multiple outputs.

Utilisation of module functions can be mixed with direct operations with the DataFields. If you do this it is important to know that module functions may modify, add, remove and also replace data items in the container. The following code snippet that runs a function and then fetches a data field

```
gwy.gwy_process_func_run('weird_func', container, gwy.RUN_IMMEDIATE)
data_field = container['/5/data']
```

is not equivalent to

```
data_field = container['/5/data']
gwy.gwy_process_func_run('weird_func', container, gwy.RUN_IMMEDIATE)
```

because the function `'weird_func'` could have replaced (or removed) the data field at `'/5/data'`. So while `data_field` will remain a valid DataField, it may no longer represent the image number five in the file.

The function name is the name of high-level "functions" provided by data processing modules. Each corresponds to some action available in the Data Process menu. If you extend Gwyddion with a module, the functions it defines will be available the same way as functions of built-in modules.

The list of functions can be obtained in several ways (apart from studying the source code):

- Using the on-line module browser. If you mouse over a module name the browser will show you the list of all provided functions. Those with type listed as "proc" are data processing functions. If you know the menu path the function can be looked up quickly by searching the expanded on-line module browser. Note that the on-line browser lists modules (and functions) available in the last stable release.
- Looking the function up within Gwyddion in *Info* → *Module Browser*. It provides the same information as the on-line browser, except that it lists modules and functions actually present in your Gwyddion installation.
- Using the log. When you use a data processing function it will appear in the log, including the names of its parameters, called settings. This is perhaps the most convenient method since you can interactively see the functions appearing in the log as you execute them.

Some functions have no adjustable parameters, for instance `"fix_zero"`. But many have some, and some have many. The last used values of all parameters are stored in settings. When a data processing function is executed it reads its parameters from the settings and, depending on the mode, either immediately performs the operation or shows a dialogue where you can adjust the values. Hence if you simply run a function from a pygwy script it will use the parameters stored in settings. This is useful if you want to use the function interactively first, find a good set of parameters and then run it many times with exactly the same settings using a script.

On the other hand, sometimes you want the script to run the function with predefined settings. In this case you can obtain the settings with

```
settings = gwy.gwy_app_settings_get()
```

It is again a Container and there are again some rules what is stored where. Module function settings are generally stored under `"/module/`*`modulename`*`"`. Module settings are not formally documented so there are essentially two ways of finding out what settings a module has: observing the log which lists all the parameters and inspecting at the `settings` file. For instance the plane levelling module can be run with masking mode set to exclude as follows (note that you can achieve the same result directly using DataField methods):

```
settings['/module/level/mode'] = int(gwy.MASKING_EXCLUDE)
gwy.gwy_process_func_run('level', container, gwy.RUN_IMMEDIATE)
```

Managing files

One of the main purposes of scripting is automation of operations that need to be carried out many times. This often includes the processing of multiple or all data in one file and processing several data files in sequence.

Concerning the processing of multiple images in one file, the data enumeration and search functions described above make easy to obtain the data you look for. A small complete sample script that plane-levels all images in a file can be written as follows:

```
container = gwy.gwy_app_data_browser_get_current(gwy.APP_CONTAINER)
for i in gwy.gwy_app_data_browser_get_data_ids(container):
    data_field = container[gwy.gwy_app_get_data_key_for_id(i)]
    coeffs = data_field.fit_plane()
    data_field.plane_level(*coeffs)
    data_field.data_changed()
```

For execution of data processing function with `gwy_process_func_run()`, the only missing piece is selecting the image in the data browser so that they will operate on the correct one. This is done by calling `gwy_app_data_browser_select_data_fiel`

```
gwy.gwy_app_data_browser_select_data_field(container, i)
```

where `i` is the image number. Similar functions exist for other data types. When you are finished with the processing it is advisable to select again the image that was selected at the beginning. If you forget tools may behave odd (at least until you select different image by switching to its data window). A small complete running a data processing function on all images in the current file can be written:

```
container = gwy.gwy_app_data_browser_get_current(gwy.APP_CONTAINER)
orig_i = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD_ID)
for i in gwy.gwy_app_data_browser_get_data_ids(container):
    gwy.gwy_app_data_browser_select_data_field(container, i)
    gwy_process_func_run('align_rows', container, gwy.RUN_IMMEDIATE)
gwy.gwy_app_data_browser_select_data_field(container, orig_i)
```

It is also simple to iterate over all open files using `gwy_app_data_browser_get_containers()`:

```
for container in gwy.gwy_app_data_browser_get_containers():
    for i in gwy.gwy_app_data_browser_get_data_ids(container):
        # Nothing new here
```

If you want to process multiple files on disk you need some means to find or identify them on the disk but that is beyond this guide (`glob` and `os` Python modules are your friends). To load an SPM file just use `gwy_file_load()`:

```
container = gwy.gwy_file_load(filename, gwy.RUN_NONINTERACTIVE)
```

This function will identify the file type and call the correct file module to load it. The resulting Container is not automatically added to the data browser. In other words, Gwyddion does not know about it and you cannot immediately use data processing module functions on it. It is your own private Container. You can add it to the data browser using

```
gwy.gwy_app_data_browser_add(container)
```

Alternatively, you can use the application-level file loading function

```
container = gwy.gwy_app_file_load(filename)
```

that not only loads the file but also adds it to the data browser. When you process many files you should also ensure they do not pile up in the memory. Function `gwy_app_data_browser_remove()` removes a Container from the data browser. To really release the resource you should also ensure the data objects are not kept around in some other structures in your script.

Note that we used `RUN_NONINTERACTIVE` instead of `RUN_IMMEDIATE` in the `gwy_file_load()` example. File functions have two possible modes, `RUN_INTERACTIVE`, which has the same meaning as for data processing functions, and `RUN_NONINTERACTIVE`, which means the function must not display any dialogues. Most file functions do not have any user interface, notable exceptions being image export and import of raw data and other types of data that cannot be loaded automatically.

File saving is quite similar:

```
gwy.gwy_file_save(container, filename, gwy.RUN_NONINTERACTIVE)
```

The file format is determined automatically from the extension. The same function can be used for image export. There are two main methods of creating a file with false-colour rendered image. The simple function `gwy_app_get_channel_thumbnail()` creates a GdkPixbuf which can be then save to a file using its `save()` method. This is most useful for thumbnails and simple preview images. If you want all the features of the image export module you can use `gwy_file_save()` to an image format (PNG, PDF, SVG, ...)

```
gwy.gwy_file_save(container, "image.png", gwy.RUN_NONINTERACTIVE)
```

Of course, similar to data processing functions, the image export will render the image according to its current settings. You can change the settings (they reside under `"/module/pixmap"`) to render the image in a predefined manner. Be warned that the image export has *lots* of settings.

We can put everything together in a simple standalone script that loads all files in the current directory matching a given pattern, finds the height channel in each, performs some levelling by calling module functions and extract a few basic statistical quantities:

```
import gwy, glob, sys

# Set up parameters for the 'align_rows' function.
settings = gwy.gwy_app_settings_get()
settings['/module/linematch/direction'] = int(gwy.ORIENTATION_HORIZONTAL)
settings['/module/linematch/do_extract'] = False
settings['/module/linematch/do_plot'] = False
settings['/module/linematch/method'] = 2

print 'Filename\tRa\tRms\tSkewness\tKurtosis'

# Go through files matching a given pattern:
for filename in glob.glob('MARENA-0??C-Maximum.0??'):
    container = gwy.gwy_file_load(filename, gwy.RUN_NONINTERACTIVE)
    gwy.gwy_app_data_browser_add(container)

    # Find channel(s) called 'Height', expecting to find one.
    ids = gwy.gwy_app_data_browser_find_data_by_title(container, 'Height')
    if len(ids) == 1:
        i = ids[0]
        # Select the channel and run some functions.
        gwy.gwy_app_data_browser_select_data_field(container, i)
        gwy.gwy_process_func_run('align_rows', container, gwy.RUN_IMMEDIATE)
        gwy.gwy_process_func_run('flatten_base', container, gwy.RUN_IMMEDIATE)
        # Extract simple statistics and print them.
        data_field = container[gwy.gwy_app_get_data_key_for_id(i)]
        avg, ra, rms, skew, kurtosis = data_field.get_stats()
        print '%s\t%g\t%g\t%g\t%g' % (filename, ra, rms, skew, kurtosis)
    else:
        sys.stderr.write('Expected one Height channel in %s but found %u.\n'
                         % (filename, len(ids)))

    gwy.gwy_app_data_browser_remove(container)
```

Writing modules in Python

Python modules are not very different from scripts. However, they must define certain functions and variables so that Gwyddion knows what kind of functionality the module provides, where to put it in the menus or how to execute it. Gwyddion recognises several kinds of modules: file, data processing (or, more precisely, image data processing), graph, volume data processing, XYZ data processing, layers and tools. The last two types are complex and cannot be currently written in Python. All the other types can.

All extension modules written in Python have to be placed in the user's directory in the `pygwy` subdirectory. This means in `~/.gwyddion/pygwy` (Unix) or `Documents and Settings\gwyddion\pygwy` (MS Windows) for Gwyddion to find them.

The variables a module has to define are more or less common to all types. For historical reasons they have all prefix `plugin_`, even though `module_` would be more appropriate:

`plugin_type` The module type, as an uppercase string. It can be one of `"FILE"`, `"PROCESS"`, `"GRAPH"`, `"VOLUME"` and `"XYZ"` that correspond to the possible types in an obvious manner.

`plugin_desc` A brief description of the function, used for tooltips for data processing-type modules, and as a file type description for file modules.

plugin_menu The menu path where the module should appear, as a string with slashes denoting submenus. The menu path is relative to the menu root for module functions of given kind. For instance data processing modules will always appear under *Data Process* and the variable should not include this part. File modules do not define menu path (if they do it is ignored).

plugin_icon Optional default icon name as a string (for putting the function to the toolbox). You can choose from Gwyddion and GTK+ stock icons. The former have names with underscores like `"gwy_level"`, the latter with dashes like `"gtk-execute"`. See the API documentations for the lists of available icons. There is no default icon.

plugin_sens Optional sensitivity flags as a combination of GwyMenuSensFlags values, such as `MENU_FLAG_DATA` or `MENU_FLAG_DATA_MASK`. The flags determine when the menu item (or toolbox button) should be available. If it is not given the default is to require the corresponding type of data to be present, for instance graph modules require a graph to exist. Usually the default works fine but it can be incorrect for instance for data synthesis modules (that do not require any data to already be loaded) or sometimes a module requires a mask, etc.

plugin_run Optional run mode flags as a combination of the flags `RUN_INTERACTIVE` and `RUN_IMMEDIATE`. This is mostly useful to indicate the function requires user input and cannot be run non-interactively. The default is that the function can be run in both modes.

All kinds of data processing modules (image, volume and XYZ) need to define function `run()` that takes two arguments, the current Container and the mode (`RUN_INTERACTIVE` or `RUN_IMMEDIATE`). If you do not care about the mode you can also define the function just with one argument. A simple complete example of data processing module follows:

```
plugin_menu = "/Pygwy Examples/Invert"
plugin_desc = "Invert values"
plugin_type = "PROCESS"

def run(data, mode):
   key = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD_KEY)
   gwy.gwy_app_undo_qcheckpoint(data, [key])
   data_field = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD)
   data_field.invert(False, False, True)
   data_field.data_changed()
```

It performs simple value inversion using the `invert()` of DataField. The call to `gwy_app_undo_qcheckpoint()` implements proper undo support. If you want to support undo then call it with the list of integer keys of all data items the function will modify *before you start modifying them*.

Graph modules differ only slightly, their `run()` function takes just a single argument, which is a Graph.

File modules have to provide several functions. If they implement file import they define `load()` and `detect_by_content()`. The loading function takes the file name (and optionally run mode) and returns either a Container representing the loaded file or None. The detection function's goal is to figure out if a file is of the format implemented by the module. It takes the file name, byte strings containing beginning and end of the file content (as they usually contain information useful for file type detection) and file size. It returns an integer score between 0 and 100 where 0 means "definitely not this file type" and 100 means "for sure this file type". No Gwyddion file module returns score larger than 100. If you really need to override a built-in module you can do it by returning a score larger than 100, but generally it is not recommended.

If the module implements file saving it has conversely define functions `save()` and `detect_by_name()`. The saving function takes a Container, file name and optional run mode and should write the file. Often you do not want or cannot save everything the currently open file contains. In this case you can obtain the current image (or other data object) using `gwy_app_data_browser_get_current()` exactly as in a data processing module and save only this image. The detection function is much simpler here because as the file is yet to be written the function can only use the file name. A simple complete example of file module, implementing both loading and saving, follows. A real module should take care of error handling which is not included here for brevity and clarity:

```
plugin_type = "FILE"
plugin_desc = "High definition stable format store (.hdsf)"

def detect_by_name(filename):
    if filename.endswith(".hdsf"):
        return 100
    return 0

def detect_by_content(filename, head, tail, filesize):
```

```
    if head.startswith("HDSF:"):
        return 100
    return 0

def load(filename, mode=None):
    f = file(filename)
    header = f.readline()
    magic, xres, yres, xreal, yreal = header.split()
    xres, yres = int(xres), int(yres)
    xreal, yreal = float(xreal), float(yreal)
    container = gwy.Container()
    data_field = gwy.DataField(xres, yres, xreal, yreal)
    data_field.set_data([float(z) for z in f])
    container['/0/data'] = data_field
    return container

def save(container, filename, mode=None):
    data_field = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD)
    if not data_field:
        return False
    f = file(filename, 'w')
    f.write('HDSF: %d %d %g %g\n'
            % (data_field.get_xres(), data_field.get_yres(),
               data_field.get_xreal(), data_field.get_yreal()))
    f.write('\n'.join('%g' % z for z in data_field))
    f.write('\n')
    f.close()
    return True
```

Pygwy modules are reloaded if they have changed since the last time they were used. So you can modify them while Gwyddion is running and it will always use the latest version of the module. They are, however, not re-registered. Meaning the file-level code is executed when the module is reloaded, but the values of `plugin_` variables only matter when the module is first loaded. Subsequent changes have no effect until you quit Gwyddion and run it again. So a file module cannot suddenly change to a graph module – which is probably a good thing.

Graphical user interface

If you want to add graphical user interface to the module you can use PyGTK together with Gwyddion's widgets for displaying the data. Note you need to import PyGTK explicitly with

```
import gtk
```

as, unlike `gwy`, it is not imported automatically. A module with an optional graphical user interface should generally define the two-argument form or `run()` and honour the mode by acting immediately or presenting the user interface. The typical structure for a module is:

```
import gtk

# Define module info...

def run(data, mode):
    if mode == gwy.RUN_INTERACTIVE:
        dialogue = gtk.Dialog(title='Title...', flags=gtk.DIALOG_MODAL,
                              buttons=(gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
                                       gtk.STOCK_OK, gtk.RESPONSE_OK))

        # Construct and show the user interface...

        response = dialogue.run()
        dialogue.destroy()
        if response != gtk.RESPONSE_OK:
            return
```

```
    # Process the data...
```

Pygwy scripts do not have a mode, they are simply executed, so while they can also present a dialogue in a similar manner if they do the dialogue will be shown always. The creation and use of checkboxes, buttons, sliders, entries and various other user interface element is beyond the scope of this guide. Please refer to PyGTK's extensive documentation.

The module can use the settings the same way as other Gwyddion modules. At the beginning, read the parameters from settings, handling the case your parameters may not be present in the setting yet:

```
settings = gwy.gwy_app_settings_get()

tuning_parameter = 1.0
if '/module/mymodule/tuning_parameter' in settings:
    tuning_parameter = settings['/module/mymodule/tuning_parameter']
```

and at the end save parameter values to the settings so that you can recall them next time:

```
settings['/module/mymodule/tuning_parameter'] = tuning_parameter
```

If you want to display image data you can use Gwyddion DataView widget. It works somewhat indirectly. Instead of giving it the DataField to display you always give it a Container and then tell it where to find the things to display. Image display is done by creating a so-called data view layer and adding it to the data view. A simple module that just displays the current image could look like:

```
import gtk, gobject

plugin_menu = "/Pygwy Examples/Display Data"
plugin_desc = "Just display the current image"
plugin_type = "PROCESS"

def run(data, mode):
    dialogue = gtk.Dialog(title='Display Data', flags=gtk.DIALOG_MODAL,
                          buttons=(gtk.STOCK_CLOSE, gtk.RESPONSE_CLOSE,))

    data_field = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD)

    container = gwy.Container()
    container['/0/data'] = data_field

    data_view = gwy.DataView(container)
    data_view.set_zoom(400.0/data_field.get_xres())
    data_view.set_data_prefix('/0/data')

    layer = gwy.LayerBasic()
    layer.set_data_key('/0/data')
    data_view.set_base_layer(layer)

    dialogue.vbox.pack_start(data_view, False, 4)

    dialogue.show_all()
    dialogue.run()
    dialogue.destroy()
```

Note the use of `set_zoom()` that limits the data view size to 400 pixels.

To let the user select points or lines interactively on the preview you need to create another layer, called vector layer and add it to the data view. Each vector layer has corresponding type of Selection that contains the coordinates of selected geometrical shapes. A simple module allowing the user to select a point on the image and displaying its coordinates could be implemented as follows:

```
import gtk, gobject

plugin_menu = "/Pygwy Examples/Analyse"
plugin_desc = "Display the current image and selected coordinates"
```

```
plugin_type = "PROCESS"

def update_info_label(selection, i, label, vformat):
    if not selection:
        label.set_markup('')
        return

    coords = selection[0]
    label.set_markup('(%.*f, %.*f) %s'
                     % (vformat.precision, coords[0]/vformat.magnitude,
                        vformat.precision, coords[1]/vformat.magnitude,
                        vformat.units))

def run(data, mode):
    dialogue = gtk.Dialog(title='Analyse Data', flags=gtk.DIALOG_MODAL,
                          buttons=(gtk.STOCK_CLOSE, gtk.RESPONSE_CLOSE,))

    data_field = gwy.gwy_app_data_browser_get_current(gwy.APP_DATA_FIELD)

    container = gwy.Container()
    container['/0/data'] = data_field

    data_view = gwy.DataView(container)
    data_view.set_zoom(400.0/data_field.get_xres())
    data_view.set_data_prefix('/0/data')

    layer = gwy.LayerBasic()
    layer.set_data_key('/0/data')
    data_view.set_base_layer(layer)

    layer = gobject.new(gobject.type_from_name('GwyLayerPoint'))
    data_view.set_top_layer(layer)
    layer.set_selection_key('/0/select/point')
    selection = layer.ensure_selection()

    dialogue.vbox.pack_start(data_view, False, 4)

    info_label = gtk.Label()
    info_label.set_alignment(0.0, 0.5)
    dialogue.vbox.pack_start(info_label, False, 4)

    value_format = data_field.get_value_format_xy(gwy.SI_UNIT_FORMAT_VFMARKUP)
    selection.connect('changed', update_info_label, info_label, value_format)

    dialogue.show_all()
    response = dialogue.run()
    dialogue.destroy()
```

Remarks

The documentation of functions available in Python is relatively complete but the price is that only a small portion of it was written specifically for the Python functions. Most of the documentation is generated from the documentation of corresponding C functions. Hence the documentation occasionally contains residue of information relevant only in C.

This includes namely ownership rules and other rules for array-like function arguments. In Python everything is passed and returned by value. In particular strings, that are not even modifiable in Python, so any comments on string ownership and modification have to be disregarded.

Gwyddion Python functions that take array-like arguments accept any Python sequence (list, tuple, array, ...) with items of the expected type (usually floats or integers). And if they return array-like data they always return newly created lists you can freely modify without influencing anything.

Typically, Python functions also do not take separate sequence-size arguments that the documentation generated from C might talk about. If there is some restriction on the number of items, it of course continues to hold in Python. For instance, the

`set_data()` method of Selection simply takes a flat sequence of coordinates. However, the sequence length must be still a multiple of "object size", i.e. the number of coordinates constituting one selected object for given selection type.

4.32 Plug-ins

Plug-ins are external programs that can be executed by Gwyddion to either perform some operation on the data or to read or write data in a third-party file format. In general, plug-ins are programs that can register themself within Gwyddion (for example printing something on standard output) to enable Gwyddion to create plug-in menu choice and can be used for data processing (or IO operation).

Generally it is preferable to extend Gwyddion functionality by modules, because modules are dynamic libraries linked directly to Gwyddion at run-time allowing much more versatile interaction with the application, and they are also faster (for the same reason). For example, plug-ins generally cannot make use of existing Gwyddion data processing functions and cannot modify data in-place, a new window is always created for the result. Programming of modules is also no harder than programming of plug-ins, maybe it is even easier (assuming you know C).

Warning The plug-in mechanism is deprecated. It will remain supported in Gwyddion 2.x, however, it will not be extended or improved. The recommended method to extend Gwyddion by routines in another language is to use language bindings, at this moment a Python interface is available. The recommended method to run third-party programs is to write a small specialized C module that knows how to communicate with these programs.

Chapter 5

Summaries and Tables

This chapter provides the reference for various formats, syntaxes and command options that you might find helpful. Most are related to advanced use of Gwyddion, except perhaps the table of common keyboard shortcuts and table of supported file formats.

5.1 gwyddion

gwyddion — SPM data visualization and analysis

Synopsis

`gwyddion` [*OPTION*...] [*FILE*...]

Description

Gwyddion is a graphical SPM (Scanning Probe Microscope) data visualization and analysis program, using GTK+.

Options

The program accepts all standard GTK+, GDK, and GtkGLExt options like `--display` or `--sync`. Please see documentation of these packages for description of toolkit options.

The behaviour of the remote control options `--remote-*` is undefined when more than one instance of Gwyddion is running on the display. They can choose an arbitrary instance to communicate to. The last remote control option given (including `--new-instance`) overrides all preceding ones.

If a directory is given as *FILE* argument the program opens a file chooser in this directory.

Gwyddion options:

`--help` Prints a brief help and terminates.

`--version` Prints version information and terminates.

`--no-splash` Disables splash screen on program startup.

`--remote-new` Opens files given on the command line in an already running instance of Gwyddion on the display. Runs a new instance if none is running.

This is probably the most useful remote control option. File type associations are usually installed to run Gwyddion with this option.

`--remote-existing` Opens files given on the command line in an already running instance of Gwyddion on the display. Fails if none is running.

This is useful if you want to handle the case of Gwyddion not running differently than by starting it.

`--remote-query` Succeeds if an instance of Gwyddion is already running on the display and prints its instance identifier. Fails if none is running.

The instance identifier depends on the remote control backend in use. In some cases it is useful as a global window identifier, in some it is not. With libXmu this option prints the X11 Window, on Win32 HWND is printed, while with LibUnique the startup id is printed.

`--new-instance` Runs a new instance of Gwyddion. It can also used to override preceding remote control options and to ensure a new instance is run when the default remote control behaviour is modified.

`--identify` Instead of running the user interface and opening *FILE*s, it detects the file type for each and terminates.

The SPM file type printed corresponds to the description shown in the list of supported file formats in the user guide. The file type is followed (in square brackets) by the name of Gwyddion file import module that would be used to load the file

and detection score. Scores considerably lower than 100 mean that although the detection produced a possible file type, it is unsure about it.

If the file type is not recognised at all, `Unknown` is printed as the file type. The program exit code is 1 if any *FILE* was not recognised.

--check Instead of running the user interface and opening *FILE*s, it loads the files, performs a sanity check on them (printing errors to standard error output) and terminates.

--convert-to-gwy=*OUTFILE.gwy* Instead of running the user interface and opening *FILE*s, it reads them, merges all the data and writes a GWY file *OUTFILE.gwy*.

--disable-gl Disables OpenGL entirely, including any checks whether it is available. This option, of course, has any effect only if Gwyddion was built with OpenGL support and one of the most visible effects is that 3D view becomes unavailable. However, you may find it useful if you encounter a system so broken that even checking for OpenGL capabilities leads to X server errors. It can also help when you run Gwyddion remotely using X11 forwarding and the start-up time seems excessively long.

--log-to-file Write messages from GLib, GTK+, Gwyddion, etc. to `~/.gwyddion/gwyddion.log` or file given in GWYDDION_LOGFILE environment variable. This option is most useful on Unix as on Win32 messages are redirected to a file by default. Logging to a file and console are not exclusive; messages can go to both.

--no-log-to-file Prevents writing messages from GLib, GTK+, Gwyddion, etc. to a file. This is most useful on Win32 where messages are written to a file by default.

--log-to-console Print messages from GLib, GTK+, Gwyddion, etc. to the console. More precisely, debugging messages are printed to the standard output, errors and warnings to the standard error. On Unix messages are printed to the console by default. Logging to a file and console are not exclusive; messages can go to both.

--no-log-to-console Disables printing messages to the console. This is most useful on Unix where messages are printed to the console by default.

--disable-modules=*MODULE,...* Prevents the registration modules of given names. This is mostly useful for development and debugging. For instance, you might want to use `--disable-modules=pygwy` when running under Valgrind for faster startup (and possibly to avoid extra errors).

--startup-time Prints wall-clock time taken by various startup (and shutdown) tasks. Useful only for developers and people going to complain about too slow startup.

Environment

On Linux/Unix, following environment variables can be used to override compiled-in installation paths (MS Windows version always looks to directories relative to path where it was installed). Note they are intended to override system installation paths therefore they are not path lists, they can contain only a single path.

GWYDDION_DATADIR Base data directory where resources (color gradients, OpenGL materials, ...) were installed. Gwyddion looks into its `gwyddion` subdirectory for resources.

When it is unset, it defaults to compiled-in value of `${datadir}` which is usually `/usr/local/share`.

GWYDDION_LIBDIR Base library directory where modules were installed. Gwyddion looks into its `gwyddion/modules` subdirectory for modules.

When it is unset, it defaults to compiled-in value of `${libdir}` which is usually `/usr/local/lib` or `/usr/local/lib64`.

GWYDDION_LIBEXECDIR Base lib-exec directory where plug-ins were installed. Gwyddion looks into its `gwyddion/plugins` subdirectory for plug-ins.

When it is unset, it defaults to compiled-in value of `${libexecdir}` which is usually `/usr/local/libexec`.

GWYDDION_LOCALEDIR Locale data directory where message catalogs (translations) were installed.

When it is unset, it defaults to compiled-in value of `${datadir}/locale` which is usually `/usr/local/share/locale`.

Other variables that influence Gwyddion run-time behaviour include GLib+ variables and GTK+ variables and some Gwyddion-specific variables:

GWYDDION_LOGFILE Name of file to redirect log messages to. On MS Windows, messages are always sent to a file as working with the terminal is cumbersome there. The default log file location, `gwyddion.log` in user's Documents and Settings, can be overridden with `GWYDDION_LOGFILE`. On Unix, messages go to the terminal by default and this environment variable has effect only if `--log-to-file` is given.

If Gwyddion is built with OpenMP support, it utilizes parallelization (not all data processing methods implement parallelization, but a sizable part does). OpenMP environment variables such as `OMP_NUM_THREADS` can be used to tune it.

Files

~/.gwyddion/settings Saved user settings and tool states. Do not edit while Gwyddion is running, it will overwrite it at exit.

~/.gwyddion/glmaterials, ~/.gwyddion/gradients, ... User directories with various resources (OpenGL materials, color gradients, ...).

$GWYDDION_DATADIR/gwyddion/glmaterials, $GWYDDION_DATADIR/gwyddion/gradients ... The same for system-wide resources.

~/.gwyddion/pixmaps Directory to place user icons to. This is mainly useful for installation of modules to home.

$GWYDDION_DATADIR/gwyddion/pixmaps, The same for system-wide icons.

~/.gwyddion/modules Directory to place user modules to. They should be placed into `file`, `graph`, `process`, `layer`, and `tools` subdirectories according to their kind, though this is more a convention than anything else.

$GWYDDION_LIBDIR/gwyddion/modules, The same for system-wide modules.

~/.gwyddion/plugins Directory to place user plug-ins to. They should be placed into `file` and `process` subdirectories according to their kind.

$GWYDDION_LIBEXECDIR/gwyddion/plugins, The same for system-wide plug-ins.

~/.gwyddion/pygwy Directory to place user python modules or scripts to.

See also

gwyddion-thumbnailer(1), gxsm(1)

5.2 gwyddion-thumbnailer

gwyddion-thumbnailer — Creates thumbnails of SPM data files

Synopsis

`gwyddion-thumbnailer` --version | --help

`gwyddion-thumbnailer` [*OPTION*...] *MODE* [*ARGUMENT*...]

Description

Gwyddion-thumbnailer creates thumbnails of SPM (Scanning Probe Microscope) image files. Depending on the mode of operation, described below, the thumbnails are written to conform to various desktop standards so that they can be displayed in nautilus(1), thunar(1) and similar file managers.

Gwyddion-thumbnailer loads and renders files using gwyddion(1), libraries and modules, therefore, it can create thumbnails of all file formats supported by your Gwyddion installation. This also means it inherits Gwyddion settings, e.g. the default false color gradient, and that it is influenced by the same environment variables as Gwyddion.

Informative Options

--help Prints a brief help and terminates.

--version Prints version information and terminates.

Thumbnailing Options

--update Writes the thumbnail only if it does not exist yet or does not seem to be up-to-date. By default, gwyddion-thumbnailer overwrites existing thumbnails with fresh ones even if they seem up to date.

Mode

Three thumbnailing modes are available: `gnome2`, `tms` and `kde4`; and one special mode: `check`. They are described below.

Gnome 2

`gwyddion-thumbnailer` [*OPTION*...] gnome2 *MAX-SIZE INPUT-FILE OUTPUT-FILE*

In `gnome2` mode, gwyddion-thumbnailer creates PNG thumbnails according to the Gnome thumbnailer specification. Using the convention from this specification, it should be run

```
gwyddion-thumbnailer gnome2 %s %i %o
```

Gwyddion installs the corresponding GConf schemas and enables thumbnailers for all file types it supports by default, so usually this should Just Work and should not need to be set up manually.

The thumbnails created in `gnome2` mode are identical as in `tms` mode, including all the PNG auxiliary chunks (provided that the same *MAX-SIZE* as in `tms` mode is specified, of course).

TMS

`gwyddion-thumbnailer` [*OPTION*...] tms *MAX-SIZE INPUT-FILE*

In `tms` mode, gwyddion-thumbnailer creates PNG thumbnails according to the Thumbnail Managing Standard. Argument *MAX-SIZE* must be `128` or `normal` (both meaning 128 pixels) or `256` or `large` (both meaning 256 pixels).

Output file name is not given as it is prescribed by the TMS. The thumbnail is placed to the directory for normal or large thumbnails according to given *MAX-SIZE*.

This mode can also be useful for manual batch-creation of thumbnails. For instance, to create them for all `*.afm` files in directory `scans` and its subdirectories, you can run

```
find scans -type f -name '*.afm' -print0 \\
    | xargs -0 -n 1 gwyddion-thumbnailer --update tms normal
```

And then go make yourself a coffee because this will take some time.

KDE 4

`gwyddion-thumbnailer` kde4 *MAX-SIZE INPUT-FILE*

In `kde4` mode, gwyddion-thumbnailer creates PNG thumbnails that are intended to be consumed by gwythumbcreator KDE module. The thumbnail, again identical as in the other modes, is written to the standard output.

Do *not* use this mode from the command line. It is documented for completeness, however, the protocol between gwythumbcreator and gwyddion-thumbnailer must be considered private and it can change at any time.

Check

`gwyddion-thumbnailer` check *INPUT-FILE*

The `check` mode does not serve for thumbnail creation. Instead, gwyddion-thumbnailer prints information about available thumbnails of *INPUT-FILE* and cached failures to produce a thumbnail by individual applications, as described by the TMS.

If the normal-sized thumbnail exists and is up to date, the large version does not exist and there is one cached failure from gnome-thumbnail-factory, the output can be for instance:

```
File:   INPUT-FILE
URI:    file:///home/me/Pictures/naughty/broken-tip3/INPUT-FILE
Normal: /home/me/.thumbnails/normal/MD5.png
        status: OK
Large:  /home/me/.thumbnails/large/MD5.png
        status: Thumbnail does not exist or stat() fails on it.
Failed: /home/me/.thumbnails/fail/gnome-thumbnail-factory/MD5.png
```

URI is the canonical URI of the input file, *MD5* stands for the hex representation of MD5 sum of the URI, as described by the TMS. If there are no cached failures, no Failed lines are printed.

This function can be used to check thumbnails of any kind, not necessarily created by gwyddion or gwyddion-thumbnailer. In future, it might be reported as an error if the thumbnail does not contain Gwyddion-specific information though.

See also

gwyddion(1),

5.3 Keyboard Shortcuts

Shortcut	Menu equivalent	Context	Action
F1		almost all windows	Open a relevant section of this user guide in a web browser.
Ctrl-Q	*File* → *Quit*	toolbox, data window, 3D window, graph window, tool window	Quit Gwyddion.
Ctrl-O	*File* → *Open*	toolbox, data window, 3D window, graph window, tool window	Open a data file.
Ctrl-S	*File* → *Save*	toolbox, data window, 3D window, graph window, tool window	Save current data (you will be prompted for a file name if none is associated yet).
Ctrl-Shift-S	*File* → *Save As*	toolbox, data window, 3D window, graph window, tool window	Save current data under a different name. The file name associated with the data changes to the new name.
Ctrl-W	*File* → *Close*	any kind of data window (image, 3D, XYZ, volume, graph), toolbox, data browser	Close file containing the current data.
Ctrl-Shift-M	*File* → *Merge*	toolbox, data window, 3D window, graph window, tool window	Merge data from a file to the current file.
Ctrl-Shift-I	*File* → *Open Image Stack*	toolbox, data window, 3D window, graph window, tool window	Load a sequence of same-sized images as volume data.
Ctrl-H	*File* → *Open Recent* → *Document History*	toolbox, data window, 3D window, graph window, tool window	Open the document history browser (or bring it forward if it is already displayed).
Ctrl-D	Duplicate in data browser	any kind of data window (image, 3D, XYZ, volume, graph), toolbox, data browser	Duplicate data in current window as new data in the same file.
Ctrl-Insert	Extract to new file in data browser	any kind of data window (image, 3D, XYZ, volume, graph), toolbox, data browser	Create new file containing just the current data.
Ctrl-Delete	Delete in data browser	any kind of data window (image, 3D, XYZ, volume, graph), toolbox, data browser	Delete the current current data from the file.
Ctrl-Z	*Edit* → *Undo*	toolbox, data window, 3D window, graph window, tool window	Undo the last processing step applied on current data.
Ctrl-Y	*Edit* → *Redo*	toolbox, data window, 3D window, graph window, tool window	Redo the last processing step applied on current data.
Ctrl-K	*Edit* → *Remove Mask*	toolbox, data window, 3D window, graph window, tool windows	Remove mask from current data window.
Ctrl-Shift-K	*Edit* → *Remove Presentation*	toolbox, data window, 3D window, graph window, tool windows	Remove presentation from current data (image) window.

Shortcut	Menu equivalent	Context	Action
F3		data window, tool window, toolbox	Show tool window if it is hidden. Hide tool window if it is shown.
Ctrl-Shift-B	*Metadata Browser* (in the context menu)	image, XYZ or volume data window	Show metadata browser for the data.
+		any kind of data window (image, 3D, XYZ, volume, graph)	Zoom current data window in.
=		any kind of data window (image, 3D, XYZ, volume, graph)	Zoom current data window in.
-		any kind of data window (image, 3D, XYZ, volume, graph)	Zoom current data window out.
Z		any kind of data window (image, 3D, XYZ, volume, graph)	Zoom current data window 1:1.
Ctrl-F	*Data Process* → *Repeat Last*	toolbox, data window, 3D window, graph window, tool windows	Repeat last data processing function with the last used parameters, on current data. Normally the operation is repeated silently, but if the processing step cannot be carried out without a human interaction, a dialog is shown.
Ctrl-Shift-F	*Data Process* → *Re-Show Last*	toolbox, data window, 3D window, graph window, tool windows	Re-show parameter dialog of the last data processing function. If the operation has no parameters to set, it is performed immediately.

You can assign your own keyboard shortcuts to all functions in the menus and it is also possible to invoke tools with keyboard shortcuts.

To change the keyboard shortcut of a menu item simply select the item using the mouse or arrow keys, press the key combination you want to assing to it and it will be immediately assigned. The shortcut must be either a special key, e.g. **F3**, or a key combination including modifiers, e.g. **Ctrl-Shift-D**. It is not possible to assign bare keys such as **Q**.

To prevent inadvertent modification of shortcuts, they can be changed only if *Edit* → *Keyboard Shortcuts* is enabled. Modifications are disabled by default which is also the recommended setting during normal use.

All keyboard shortucts are stored in file `ui/accel_map` in the user's directory, which usually means `~/.gwyddion` (Unix) or `Documents and Settings\gwyddion` (MS Windows). Assigning shortcuts to tools can be only done by editing this file. Each line corresponds to an action that can be invoked with a shortcut. For instance the Mask Editor tool's line is by default:

```
; (gtk_accel_path "<tool>/GwyToolMaskEditor" "")
```

Semicolons represents comments, i.e. lines starting with a semicolon are inactive. Hence, to assign the combo **Ctrl-Shift-E** to the Mask Editor tool, remove the semicolon to make the line active and fill the desired shortcut in the empty quotes:

```
(gtk_accel_path "<tool>/GwyToolMaskEditor" "<Control><Shift>e")
```

5.4 Supported File Formats

File Format	Extensions	Module	Read	Write	SPS	Volume	Curve map
Accurion ASCII exported scanning ellipsometry	.txt	accurell	Yes	—	—	—	—
Accurex II text data	.txt	accurexii-txt	Yes	—	—	—	—
AFM Workshop spectroscopy	.csv	afmw-spec	—	—	Yes	—	—
AIST-NT	.aist	aistfile	Yes	—	—	—	—
Alicona Imaging AL3D data	.al3d	alicona	Yes	—	—	—	—
Ambios AMB	.amb	ambfile	Yes[a]	—	—	—	—
Ambios 1D profilometry data	.dat, .xml	ambprofile	Yes	—	—	—	—
Analysis Studio XML	.axd, .axz	anasys_xml	Yes	—	Yes	—	—
Anfor SIF	.sif	andorsif	Yes[a]	—	—	—	—
Anfatec	.par, .int	anfatec	Yes	—	—	—	Yes
A.P.E. Research DAX	.dax	apedaxfile	Yes	—	—	—	—
A.P.E. Research APDT	.apdt	apedaxfile	Yes	—	—	—	—
A.P.E. Research DAT	.dat	apefile	Yes	—	—	—	—
Asylum Research ARDF	.ardf	ardf	Yes	—	—	Yes	—
Curve map exported to text	.txt	asciicmap	Yes	Yes	—	—	—
Text matrix of data values	.txt	asciiexport	—	Yes	—	—	—
ASD high-speed AFM files	.asd	asdfile	Yes	—	—	—	—
Assing AFM	.afm	assing-afm	Yes	Yes	—	—	—
Attocube Systems ASC	.asc	attocube	Yes	—	—	—	—
Image Metrology BCR, BCRF	.bcr, .bcrf	bcrfile	Yes	—	—	—	—
Burleigh BII	.bii	burleigh_bii	Yes	—	—	—	—
Burleigh IMG v2.1	.img	burleigh	Yes	—	—	—	—
Burleigh exported data	.txt, .bin	burleigh_exp	Yes	—	—	—	—
Code V interferogram data	.int	codevfile	Yes	—	—	—	—
Createc DAT	.dat	createc	Yes	—	—	—	—
Benyuan CSM	.csm	csmfile	Yes	—	—	—	—
Dektak OPDx profilometry data	.OPDx	dektakvca	Yes	—	—	—	—
Dektak XML profilometry data	.xml	dektakxml	Yes	—	—	—	—
Veeco Dimension 3100D	.001, .002, etc.	dimensionfile	Yes[a]	—	—	—	—
Digital Micrograph DM3 TEM data	.dm3	dm3file	Yes	—	—	—	—
Digital Micrograph DM4 TEM data	.dm4	dm3file	Yes	—	—	—	—
DME Rasterscope	.img	dmefile	Yes	—	—	—	—

File Format	Extensions	Module	Read	Write	SPS	Volume	Curve map
Gwyddion dumb dump data	.dump	dumbfile	Yes	—	—	—	—
ECS	.img	ecsfile	Yes	—	—	—	—
Evovis XML profilometry data	.xml	evovisxml	Yes	—	—	—	—
Nanosurf EZD, NID	.ezd, .nid	ezdfile	Yes	—	—	—	—
FemtoScan SPM	.spm	femtoscan	Yes	—	—	—	—
FemtoScan text data	.txt	femtoscan-txt	Yes[b]	—	—	—	—
Flexible Image Transport System (FITS)	.fits, .fit	fitsfile	Yes	—	—	—	—
VTK structured grid file	.vtk	formats3d	—	Yes	—	—	—
PLY 3D Polygon File Format	.ply	formats3d	—	Yes	—	—	—
Wavefront OBJ 3D geometry	.obj	formats3d	Yes	Yes	—	—	—
Object File Format 3D geometry	.off	formats3d	—	Yes	—	—	—
Stereolitography STL 3D geometry (binary)	.stl	formats3d	Yes	Yes	—	—	—
XYZ data	.xyz, .dat	formats3d	Yes	Yes	—	—	—
DME GDEF	.gdf	gdeffile	Yes	—	—	—	—
Gwyddion Simple Field	.gsf	gsffile	Yes	Yes	—	—	—
Gwyddion native data	.gwy	gwyfile	Yes	Yes	Yes	Yes	—
Gwyddion XYZ data	.gxyzf	gxyzffile	Yes	Yes	—	—	—
Psi HDF4	.hdf	hdf4file	Yes	—	—	—	—
Hitachi AFM	.afm	hitachi-afm	Yes	—	—	—	—
Hitachi S-3700 and S-4800 SEM data	.txt + image	hitachi-sem	Yes	—	—	—	—
WaveMetrics IGOR binary wave v5	.ibw	igorfile	Yes	Yes	—	—	—
WaveMetrics IGOR Packed files	.pxp	igorfile	Yes	—	—	—	—
IntelliWave ESD	.esd	intelliwave	Yes	—	—	—	—
Intematix SDF	.sdf	intematix	Yes	—	—	—	—
ISO 28600:2011 SPM data transfer format	.spm	iso28600	Yes	Yes	Limited[c]	—	—
JEOL	.tif	jeol	Yes	—	—	—	—
JEOL TEM image	.tif	jeoltem	Yes[a]	—	—	—	—
JPK Instruments	.jpk, .jpk-qi-image, .jpk-force, .jpk-force-map, .jpk-qi-data .map	jpkscan	Yes	—	Yes	—	Yes
JEOL JSPM	.tif	jspmfile	Yes	—	—	—	—
Keyence profilometry VK3, VK4, VK6, VK7	.vk3, .vk4, .vk6, .vk7	keyence	Yes	—	—	—	—
Leica LIF Data File	.lif	leica	Yes	—	—	Yes	—
Olympus LEXT 4000	.lext	lextfile	Yes	—	—	—	—
FEI Magellan SEM images	.tif	magellan	Yes	—	—	—	—
MapVue	.map	mapvue	Yes	—	—	—	—

File Format	Extensions	Module	Read	Write	SPS	Volume	Curve map
Matlab MAT 5 files	.mat	matfile	Yes	—	—	—	—
Zygo MetroPro DAT	.dat	metropro	Yes	—	—	—	—
MicroProf TXT	.txt	microprof	Yes	—	—	—	—
MicroProf FRT	.frt	microprof	Yes	—	—	—	—
DME MIF	.mif	miffile	Yes	—	—	—	—
Molecular Imaging MI	.mi	mifile	Yes	—	Limited[c]	—	Yes
Aarhus MUL	.mul	mulfile	Yes	—	—	—	—
Nanoeducator	.mspm, .stm, .spm	nanoeducator	Yes	—	Yes	—	—
Nanomagnetics NMI	.nmi	nanomagnetics	Yes	—	—	—	—
Nanonics NAN	.nan	nanonics	Yes	—	—	—	—
Nanonis SXM	.sxm	nanonis	Yes	—	—	—	—
Nanonis STS spectroscopy	.dat	nanonis-spec	—	—	Yes	—	—
Nano-Solution/NanoObserver	.nao	nanoobserver	Yes	—	Yes	—	—
Nanoscan XML	.xml	nanoscan	Yes	—	—	—	—
NanoScanTech	.nstdat	nanoscantech	Yes	—	Yes	Yes	—
Veeco Nanoscope III	.spm, .001, .002, etc.	nanoscope	Yes	—	Limited[c]	Yes	Yes
Veeco Nanoscope II	.001, .002, etc.	nanoscope-ii	Yes	—	—	—	—
NanoSystem profilometry	.spm	nanosystemz	Yes	—	—	—	—
Nanotop SPM	.spm	nanotop	Yes	—	—	—	—
GSXM NetCDF	.nc	netcdf	Yes	—	—	—	—
Nano Measuring Machine profile data	.dsc + .dat	nmmxyz	Yes[d]	—	—	—	—
Nova ASCII	.txt	nova-asc	Yes	—	—	—	—
Numpy binary serialised array	.npy	npyfile	Yes	—	—	—	—
Multiple numpy binary serialised arrays	.npz	npyfile	Yes	—	—	—	—
Nearly raw raster data (NRRD)	.nrrd	nrrdfile	Yes[e]	Yes[f]	—	Yes	—
NT-MDT	.mdt	nt-mdt	Yes	—	Yes	Yes	—
EM4SYS NX II	.bmp	nxiifile	Yes	—	—	—	—
Olympus OIR	.oir	oirfile	Yes	—	—	—	—
Olympus packed OIR	.poir	oirfile	Yes	—	—	—	—
NT-MDT old MDA	.sxml + .dat	oldmda	—	—	—	Yes	—
Olympus LEXT 3000	.ols	ols	Yes	—	—	—	—
Open Microscopy OME TIFF	.ome.tiff, .ome.tif	ometiff	Yes	—	—	—	—
Omicron SCALA	.par + .tf*, .tb*, .sf*, .sb*	omicron	Yes	—	Yes	—	—
Omicron flat format	.*_flat	omicronflat	Yes	—	—	—	—
Omicron MATRIX	.mtrx	omicronmatrix	Yes	—	Limited[c]	Yes	—
Wyko OPD	.opd	opdfile	Yes	—	—	—	—
Wyko ASCII	.asc	opdfile	Yes	—	—	—	—
OpenGPS X3P (ISO 5436-2)	.x3p	opengps	Yes	—	—	—	—

File Format	Extensions	Module	Read	Write	SPS	Volume	Curve map
NASA Phoenix Mars mission AFM data	.dat, .lbl + .tab	phoenix	Yes	—	—	—	—
Pixmap images	.png, .jpeg, .tiff, .tga, .pnm, .bmp	pixmap	Yes[g]	Yes[h]	—	—	—
Nanosurf PLT	.plt	pltfile	Yes	—	—	—	—
Pacific Nanotechnology PNI	.pni	pnifile	Yes	—	—	—	—
Princeton Instruments camera SPE	.spe	princetonspe	Yes	—	—	—	—
Park Systems	.tiff, .tif	psia	Yes	—	—	—	—
Park Systems PS-PPT	.ps-ppt	psppt	Yes	—	—	—	Yes
SymPhoTime TTTR v2.0 data	.pt3	pt3file	Yes	—	—	—	—
Quazar NPIC	.npic	quazarnpic	Yes[a]	—	—	—	—
Quesant AFM	.afm	quesant	Yes	—	—	—	—
Raw text files	any	rawfile	Yes	—	—	—	—
Raw binary files	any	rawfile	Yes	—	—	—	—
Graph text data (raw)	any	rawgraph	Yes[i]	—	—	—	—
Renishaw WiRE Data File	.wdf	renishaw	Yes	—	Yes	Yes	—
RHK Instruments SM3	.sm3	rhk-sm3	Yes	—	Limited[c]	—	—
RHK Instruments SM4	.sm4	rhk-sm4	Yes	—	Limited[c]	—	—
RHK Instruments SM2	.sm2	rhk-spm32	Yes	—	Limited[c]	—	—
RMI TMD 3D measurement	.tmd	rmitmd	Yes	—	—	—	—
Automation and Robotics Dual Lens Mapper	.mcr, .mct, .mce	robotics	Yes	—	—	—	—
S94 STM files	.s94	s94file	Yes	—	—	—	—
Cyber technologies SCAN data	.scan	scanfile	Yes	—	—	—	—
Cyber technologies SCNX data	.scnx	scnxfile	Yes	—	—	—	—
Surfstand Surface Data File	.sdf	sdfile	Yes	Yes	—	—	—
Micromap SDFA	.sdfa	sdfile	Yes	—	—	—	—
Seiko SII	.xqb, .xqd, .xqt, .xqp, .xqj, .xqi	seiko	Yes	—	—	—	—
Sensofar PLu	.plu, .apx	sensofar	Yes	—	—	—	—
Sensofar PLUx data	.plux	sensofarx	Yes	—	—	—	—
Sensolytics DAT	.dat	sensolytics	Yes	—	—	—	Yes
Shimadzu	.sph, .spp, .001, .002, etc.	shimadzu	Yes	—	—	—	—
Shimadzu ASCII	.txt	shimadzu	Yes	—	—	—	—
IonScope SICM	.img	sicmfile	Yes	—	—	—	—
Surface Imaging Systems	.sis	sis	Yes	—	—	—	—
Thermo Fisher SPC File	.spc	spcfile	—	—	Yes	—	—
SPIP ASCII	.asc	spip-asc	Yes	Yes	—	—	—

File Format	Extensions	Module	Read	Write	SPS	Volume	Curve map
Thermicroscopes SPM-Lab R4-R7	.tfr, .ffr, etc.	spmlab	Yes	—	—	—	—
Thermicroscopes SPM-Lab floating point	.flt	spmlabf	Yes	—	—	—	—
SPML (Scanning Probe Microscopy Markup Language)	.xml	spml	Yes	—	—	—	—
ATC SPMxFormat data	.spm	spmxfile	Yes	—	—	—	—
Omicron STMPRG	tp*, ta*	stmprg	Yes	—	—	—	—
Molecular Imaging STP	.stp	stpfile	Yes	—	—	—	—
Surf	.sur	surffile	Yes	—	—	—	—
ThermoFisher Talos TEM images	.tif	talos	Yes	—	—	—	—
Tescan MIRA SEM images	.tif	tescan	Yes	—	—	—	—
Tescan LYRA SEM images	.hdr + .png	tescan	Yes	—	—	—	—
FEI Tecnai imaging and analysis (former Emispec) data	.ser	tiaser	Yes	—	Yes	Yes	—
Corning Tropel UltraSort topograhpical data	.ttf	ttffile	Yes	—	—	—	—
Corning Tropel exported CSV data	.csv	ttffile	Yes	—	—	—	—
Unisoku	.hdr + .dat	unisoku	Yes	—	—	—	—
WinSTM data	.stm	win_stm	Yes	—	—	—	—
WITec Project data	.wip	wipfile	Yes	—	Yes	Yes	—
WITec ASCII export	.dat	witec-asc	Yes	—	—	—	—
WITec	.wit	witfile	Yes	—	—	—	—
Department of Nanometrology, WRUST	.dat	wrustfile	Yes	—	—	—	—
AFM Workshop data	.wsf	wsffile	Yes	—	—	—	—
Nanotec WSxM	.tom, .top, .stp	wsxmfile	Yes	Yes	—	—	—
Nanotec WSxM curves	.cur	wsxmfile	Yes	—	—	—	—
Carl Zeiss SEM scans	.tif	zeiss	Yes	—	—	—	—
Carl Zeiss CLSM images	.lsm	zeisslsm	Yes	—	—	Yes	—
Zemax grid sag data	.dat	zemax	Yes	—	—	—	—
KLA Zeta profilometry data	.zmg	zmgfile	Yes	—	—	—	—
Keyence ZON data	.zon	zonfile	Yes	—	—	—	—
ZyVector Scanz	.zad	zyvex	Yes	—	—	—	—
OpenEXR images	.exr	hdrimage	Yes	Yes	—	—	—
Zygo DATX HDF5	.datx	datxfile	Yes	—	—	—	—
EPFL HDF5	.h5	epflfile	Yes	—	—	—	—
Asylum Research Ergo HDF5	.h5,.aris	ergofile	Yes	—	—	—	—
Generic HDF5 files	.h5	hdf5file	Yes	—	—	—	—

File Format	Extensions	Module	Read	Write	SPS	Volume	Curve map
Matlab MAT 7.x files	.mat	hdf5file	Yes	—	—	—	—
Nanosurf NHF	.nhf	nhffile	Yes	—	—	—	—
N-Dimensional Spectroscopy and Imaging Data (NSID) HDF5	.h5	nsidfile	Yes	—	—	—	—
Shilps Sciences Lucent HDF5	.h5	shilpsfile	Yes	—	Yes	Yes	Yes

[a] The import module is unfinished due to the lack of documentation, testing files and/or people willing to help with the testing. If you can help please contact us.

[b] Regular sampling in both X and Y direction is assumed.

[c] Spectra curves are imported as graphs, positional information is lost.

[d] XYZ data are interpolated to a regular grid upon import.

[e] Not all variants are implemented.

[f] Data are exported in a fixed attached native-endian float point format.

[g] Import support relies on Gdk-Pixbuf and hence may vary among systems.

[h] Usually lossy, intended for presentational purposes. 16bit grayscale export is possible to PNG, TIFF and PNM.

[i] At present, only simple two-column data, imported as graph curves, are supported.

5.5 High-Depth Image Formats

Gwyddion can export data to 16bit greyscale PNG, PNM and TIFF images and to OpenEXR images with half, float and 32bit data types. In case of 16bit images the full data range is always stretched to the full greyscale range; OpenEXR export permits to specify the value scaling factor.

When data are exported to a high-depth image additional information is stored to the file to enable automated loading back to Gwyddion without having to specify the dimensions and scales manually. By storing this additional information to image files you create in other programs, you can also make them directly loadable to Gwyddion with correct dimensions and scales. The information is organised as key-value pairs, stored using individual format-specific means for each format, described in the following table.

Format	Method
PNG	`tEXt` chunks
OpenEXR	named attributes
PNM	header comments of the form `# key: value`

Most keys are identical to those used in Gwyddion Simple Fields, except for the added `Gwy::` prefix, so see also GSF description for more details. Floating point values are stored directly if the format permits it (OpenEXR), otherwise a text representation of the number is used (in the C format). The keys are listed below.

Key	Type	Meaning
`Gwy::XReal`	floating point	Horizontal size in physical units (given by `XYUnits`), a positive floating point number.
`Gwy::YReal`	floating point	Vertical size in physical units (given by `XYUnits`), a positive floating point number.
`Gwy::XOffset`	floating point	Horizontal offset in physical units (given by `XYUnits`).
`Gwy::YOffset`	floating point	Vertical offset in physical units (given by `XYUnits`).
`Gwy::ZScale`	floating point	Value scaling factor. Image data are to be multiplied by this factor to obtain physical values. This parameter is usually used with limited-range floating point formats such as half. For integer data, `Gwy::ZMin` and `Gwy::ZMax` is usually used.

Key	Type	Meaning
`Gwy::ZMin`	floating point	Value in physical units corresponding to the minimum value representable in the image (normally 0).
`Gwy::ZMax`	floating point	Value in physical units corresponding to the maximum value representable in the image.
`Gwy::XYUnits`	string	Lateral units, i.e. units of physical sizes and offsets.
`Gwy::ZUnits`	string	Value units, i.e. units of data values.
`Gwy::Title`	string	Data/channel title.

In case of PNG, the scaling information is also stored in the standard `sCAL` and `pCAL` chunks (with linear scaling formula). Conversely, if these chunks are present (and the Gwyddion-specific are absent) the information from them is used in import. See the PNG specifiation for the chunk description.

5.6 Expressions

Expressions used in Data Arithmetic module, grain quantity formulas and in graph function fitting have syntax similar to common programming languages.

All numbers are real (floating point), number literals use standard notation. Examples of valid numbers: `1`, `.707`, `2.661`, `8.2e-34`.

Function, constant, and variable names start with a letter and continue with zero or more letters, numbers, or underscores. Examples of valid identifiers: `pow10` (a function), `Pi` (a constant), `d2_2` (a variable).

The precedence of operations is summarized in following table.

Operation	Associativity	Examples
parentheses	N.A.	`(x)`
function call and unary operators	right to left	`-sqrt 3`
power operator	right to left	`2^16`
multiplication, division, and modulo operators	left to right	`9/2 * 8`
addition and subtraction operators	left to right	`3 - 4 + 5`

Note `-3^2` is 9, that is `(-3)^2`, like in **bc**, but unlike in Perl or Python.

Available operators and functions are listed in following table.

Operator	Meaning
+ (unary)	no op
− (unary)	negative value
~	negative value (equivalent to −)
+ (binary)	addition
− (binary)	subtraction
*	multiplication
/	division
%	floating point modulo
^	power
abs	absolute value
floor	rounding down to nearest integer
ceil	rounding up to nearest integer
sqrt	square root
cbrt	cubic root
sin	sine function
cos	cosine function
tan	tangent function
asin	arc sine function
acos	arc cosine function
atan	arc tangent function
sinc	cardinal sine function, sine divided by the value
exp	base-e exponential function
ln	base-e logarithm function
log	base-e logarithm function
pow10	base-10 exponential function
log10	base-10 logarithm function
pow2	base-2 exponential function
log2	base-2 logarithm function
spow	signed power function; the result has the same sign as the argument
sinh	hyperbolic sine function
cosh	hyperbolic cosine function
tanh	hyperbolic tangent function
asinh	inverse hyperbolic sine function
acosh	inverse hyperbolic cosine function
atanh	inverse hyperbolic tangent function
erf	error function (integral of Gaussian from zero)
erfc	complementary error function (integral of Gaussian to infinity)
pow	power function, `pow(x,y)` equals to `x^y`
min	minimum of two values
max	maximum of two values
step	zero for negative values or zero, one for positive values
mod	floating point modulo, `mod(x,y)` equals to `x % y`
hypot	Euclidean distance function, `hypot(x,y)` equals to `sqrt(x^2+y^2)`
atan2	arc tangent function of two variables

The following functions are available if the system mathematical library provides them:

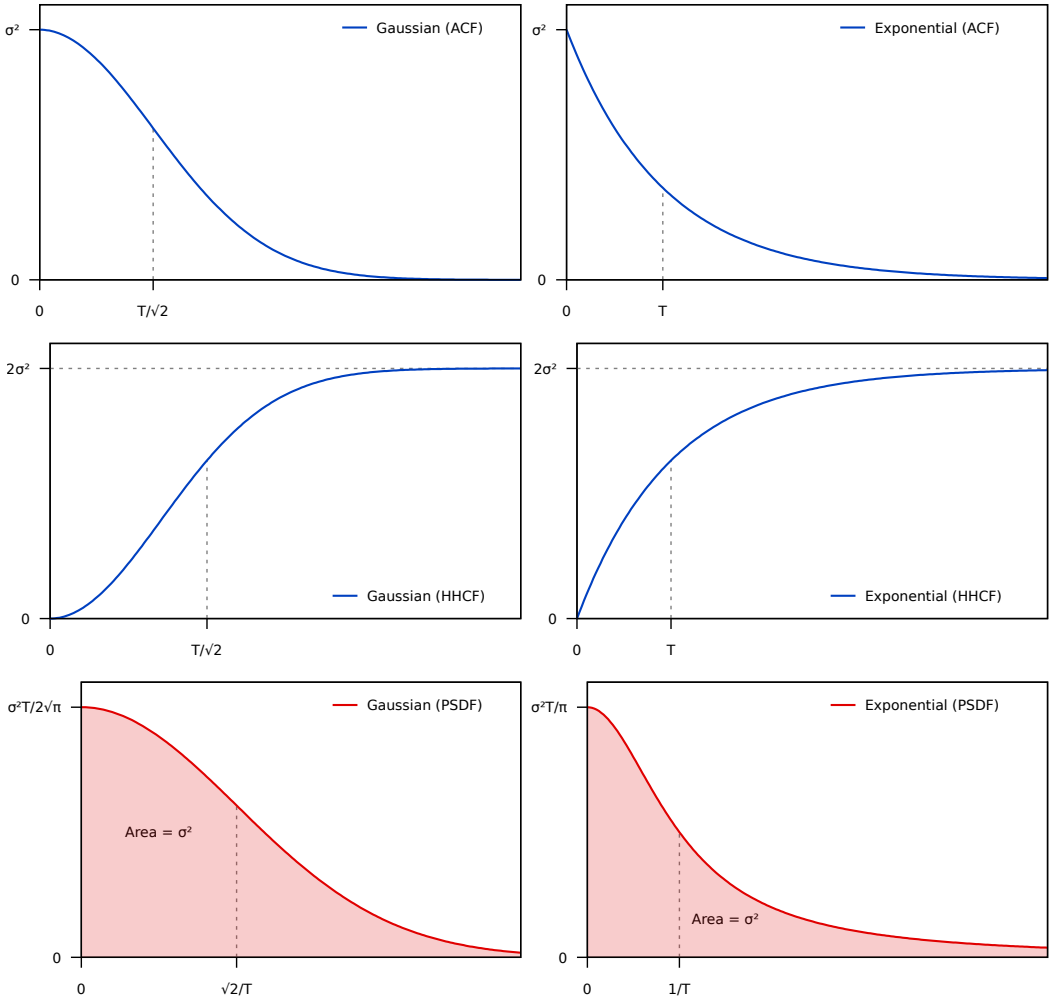
Operator	Meaning
`lGamma`	logarithm of Γ function
`Gamma`	Γ function
`J0`	Bessel function of first kind and order 0
`J1`	Bessel function of first kind and order 1
`Y0`	Bessel function of second kind and order 0
`Y1`	Bessel function of second kind and order 1

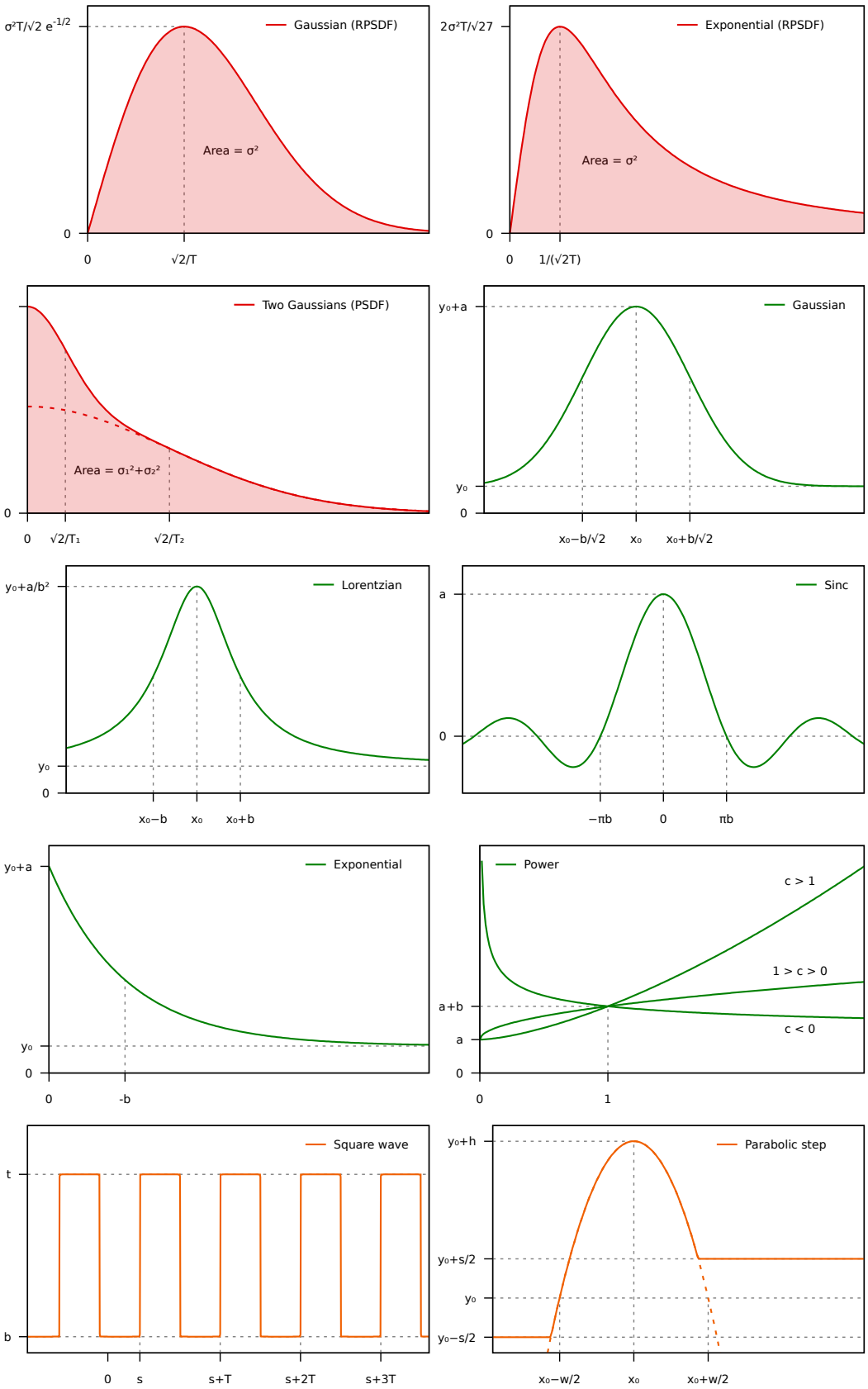
Beside that, there are a few peculiarities that may make typing simple expression easier:

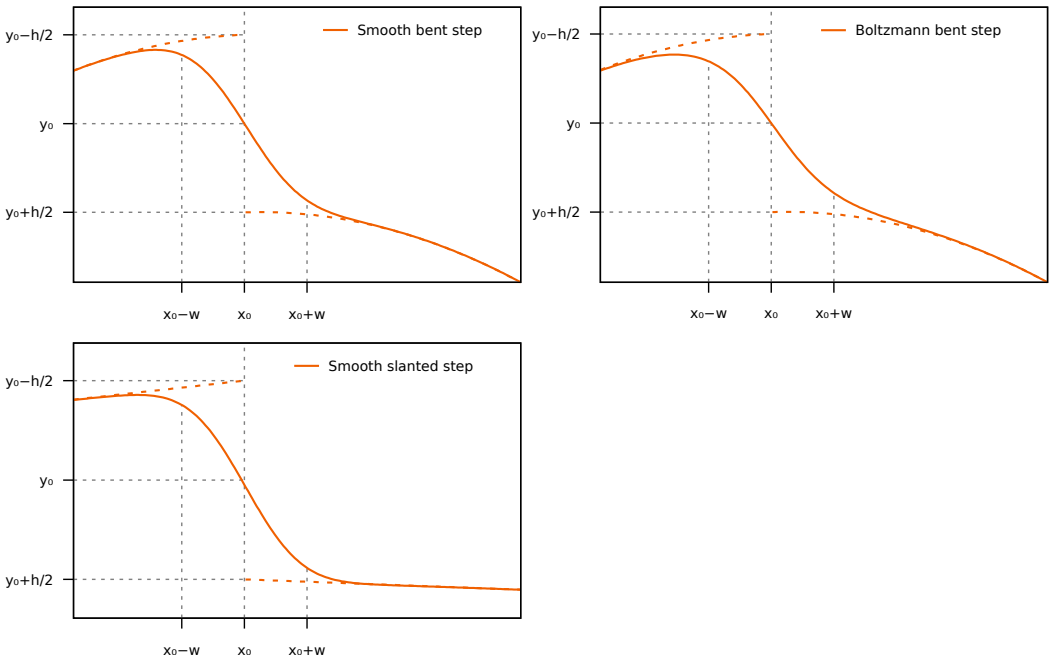
- Multiplication signs are optional, you can use spaces instead (or nothing, in some cases). E.g., `3/4 Pi` and `5(4+3)(2+1)` are valid expressions. However, `3a` is not a valid expression, `3e-4` always means `0.0003`, not `3*e - 4`.
- There is no difference between function calls and unary operators, so parentheses can be often omitted. E.g, `sqrt 5` and `hypot hypot 3,4,5` are valid expression. The latter can be parenthesized as follows: `hypot(hypot(3,4),5)`. Note however, function calls have higher priority than any other operator, thus `sin Pi/2` is the same as `(sin Pi)/2`, not as `sin(Pi/2)`.

If in doubt, write out expressions in full form.

5.7 Fitting Functions




σ²T/√2 e^-1/2
Gaussian (RPSDF)
Area = σ²
0
0
√2/T
2σ²T/√27
Exponential (RPSDF)
Area = σ²
0
0
1/(√2T)
Two Gaussians (PSDF)
Area = σ₁²+σ₂²
0
0
√2/T₁
√2/T₂
y₀+a
Gaussian
y₀
0
x₀−b/√2
x₀
x₀+b/√2
y₀+a/b²
Lorentzian
y₀
0
x₀−b
x₀
x₀+b
a
Sinc
0
−πb
0
πb
y₀+a
Exponential
y₀
0
0
-b
Power
c > 1
1 > c > 0
c < 0
a+b
a
0
0
1
t
Square wave
b
0
s
s+T
s+2T
s+3T
y₀+h
Parabolic step
y₀+s/2
y₀
y₀−s/2
x₀−w/2
x₀
x₀+w/2



5.8 Fitting 3D Shapes

Many three-dimensional geometrical shapes that can be fitted by the image and XYZ *Fit Shape* functions have common parameters. Therefore, we first describe the shared parameters and only list the specific parameters in the descriptions of individual fitable geometrical shapes.

General common parameters

x_0 Horizontal position of the feature centre, usually coinciding with the apex (or the deepest point for negative-height features). For periodic shapes it is offset of one of the infinitely many features from the coordinate origin.

y_0 Vertical position of the feature centre, usually coinciding with the apex (or the deepest point for negative-height features). For periodic shapes it is offset of one of the infinitely many features from the coordinate origin.

z_0 Height of the base plane (at the feature centre if the base plane is inclined). Note that this parameter may be 100% correlated with other height-related parameters if a feature covers the entire image and the base plane is not visible.

h Height of the feature with respect to the flat base plane.

φ Rotation in the positive direction with respect to the basic orientation, which is usually with major axes of the shape aligned horizontally and vertically.

Common parameters for bumps

b_x Slope of the base plane in the horizontal direction.

b_y Slope of the base plane in the vertical direction.

a Anisotropy parameter. It is equal to the ratio of the longer and shorter sides (or half-axes or other measures of width) of the feature. Therefore, the one dimension is equal to the mean dimension multiplied by $\sqrt{a}$ and the other to the mean dimension divided by $\sqrt{a}$. Fix a to unity if the shape should not be elongated in one direction and contracted in another.

Bump-type shapes include base plane parameters b_x and b_y and generally some parameter desribing the size of the shape at the base. If only the apex is present in the image, their values are undefined or they cease to be independent. This may cause the fit to converge poorly or fail. The parameter values need to be fixed in such case.

Common parameters for smooth bumps

The Gaussian-smoothed bumps – cone and pyramids – have the following two common parameters:

R Radius of curvature at the apex.

α Tangent of the slope of the side.

Common parameters for steps

h Height of the step above the base plane.

δ Smoothness, defining the width of error function-shaped edge between lower and upper planes.

Common parameters for periodic shapes

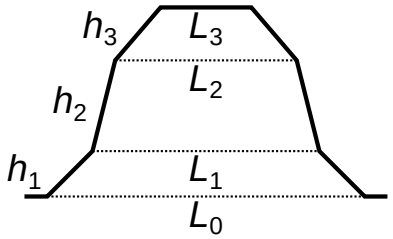
L Period, i.e. the distance after which the structure starts repeating.

p Coverage, i.e. the portion of the plane where there is some feature, as opposed to the base plane.

Specific parameters

Grating (simple) has only a single additional parameter controlling the shape, c. The ridge has the form of truncated hyperbolic cosine, with small c corresponding to a parabolic cross-section while large c corresponding to sharp, almost rectangular cross-section.

Grating (3-level) has a cross-section of three trapezoids stacked on top of another. The corresponding three height parameters h_1, h_2 and h_3 determine the heights of the individual trapezoids. The corresponding width reduction parameters q_1, q_2 and q_3 determine how much the upper base of the corresponding trapezoid is reduced with respect to the lower base (by the factor $1/(1+q_i)$). It is often useful to look at the corresponding derived quantities L_0, L_1, L_2 and L_3 that are directly equal to the widths and derived quantity h equal to the total height.



Widths and heights of the trapezoids forming the 3-level grating profile. Note that the widths are derived quantities, not direct fitting parameters.

Holes have parameters s which controls the tapering of the hole due to sloped sides and r which controls the radius of the rounder corners. For both parameters zeros correspond to an ideal rectangular shape.

Parabolic ring is a three-dimensional rotationally symmetric version of the *Parabolic step* fitting function and has exactly the same shape parameters.

Sphere has only one shape parameter C defining its curvature. The radius is available as a derived quantity.

Cylinder (lying) also has a curvature parameter C. In addition it has a inclination parameter $b_{\|}$ determining the base plane inclination along the cylinder length.

Gaussian width is controlled by the mean rms parameter σ_{mean}. For anisotropic Gaussians it is equal to the geometric mean of the two widths in the orthogonal directions that are available as derived quantities.

Lorentzian mean half width at half maximum is β_{mean}. For anisotropic Lorentzian it is equal to the geometric mean of the two widths in the orthogonal directions that are available as derived quantities.

Cone parameter R determines the cone radius at base.

Pyramid (diamond) and *Pyramid (3-sided)* parameter L determines the pyramid side at base. Since the basic orientation of this pyramid is diagonal all four sides still have the same length even when the anisotropy parameter a is not unity.

Step (two-sided) has parameter w defining the step width.

5.9 Resources

Various bits of data, e.g. false color maps or raw file import presets, are stored in standalone files that are collectively called resource files. Gwyddion looks for resources in two different locations: system and user-specific.

System resources are installed along with the program and they are not modifiable. Typically, they are located under a directory such as `/usr/share/gwyddion` (Unix), `Program Files\Gwyddion` (MS Windows) or other directory determined by GWYDDION_DATADIR.

User resources are located in a user's directory, this usually means under `~/.gwyddion` (Unix) or `Documents and Settings\gwydd` (MS Windows).

All resource files are simple text files that can be easily examined and modified by text editors or sent to other users (if they are copied or created manually Gwyddion needs to be restarted to notice them). In most cases only characters of the ASCII can appear in the files. If international text can appear there it must be in the UTF-8 encoding. Numerical values are represented in the standard POSIX format, i.e. with decimal point, independently on what decimal separator is usual in the user's language.

Resources are organized in subdirectories according to their kind, e.g. color gradients reside in the subdirectory `gradients`. The name of the file determines the resource name – gradient Gray is found in file `gradients/Gray`. Modules can define their own resource types; the types described here are the most important types but the list may not be comprehensive.

Every resource file has the same structure. It starts with a line identifying the resource type:

```
Gwyddion resource GwyGradient
```

where GwyGradient is the type name in the type system (which is quite a low-level detail but so it is), followed by named parameters in the form

```
name value
```

and resource data. Some resource types may contain only named parameters, other may contain only data.

Gradients

Gradients, i.e. false color maps, reside in directory `gradients`, they are identified by GwyGradient and contain only data. They can be edited in the application using the gradient editor.

The gradient data consists of rows corresponding to individual points in the gradient:

```
position red green blue alpha
```

The position determines where the color defined by *red*, *green*, *blue* and *alpha* components is placed in the interval $[0, 1]$ where 0 corresponds to the gradient start, 1 corresponds to the end. The color is interpolated linearly between the specified points.

The positions must form an increasing sequence from 0 to 1 (i.e. the minimum number of color points is two). The range of the color components is also $[0, 1]$. Note the alpha value, corresponding to opacity, is unused and must be given as 1 (fully opaque).

For instance, the standard gradient Red going from black (0 0 0) to red (1 0 0) to white (1 1 1) is defined as follows:

```
Gwyddion resource GwyGradient
0.0 0 0 0 1
0.5 1 0 0 1
1.0 1 1 1 1
```

OpenGL Materials

OpenGL materials reside in directory `glmaterials`, they are identified by GwyGLMaterial and contain only data. They can be edited in the application using the OpenGL material editor.

The material data consists of four RGBA lines, similar to gradients that correspond in to the four OpenGL material components in the following order:

1. ambient,
2. diffuse,
3. specular,

4. emission.

See section OpenGL Material Editor for explanation of the components. They are followed by a line containing the shininess, again as a number from the interval $[0, 1]$.

Note the emission component, while read and written by Gwyddion, is presently unused by the 3D view. It is recommended to set it to 0 0 0 1, i.e. black.

For instance, the standard material Red-Rubber with very dark red ambient color, grayish diffuse reflection, red specular reflection and low shininess is defined as follows:

```
Gwyddion resource GwyGLMaterial
0.05 0.0  0.0  1.0
0.5  0.4  0.4  1.0
0.7  0.04 0.04 1.0
0.0  0.0  0.0  1.0
.078125
```

Grain Values

Grain values reside in directory `grainvalues`, they are identified by GwyGrainValue and contain only named parameters. They can be used to define additional grain quantities, derived from the built-in quantities, that appear under *User* group in grain analysis functions. At the time of writing this, there is no editor in the application, new quantities must be created manually.

The named parameters are summarized in the following table:

Parameter	Required	Type	Description
`symbol`	required	identifier	Identifier to use in other expressions (but see below). It must be a valid identifier of ASCII letters, numbers and underscores, starting with a letter.
`expression`	required	free-form	Formula for calculation of this quantity from other grain quantities. The general expression syntax is described in section Expressions.
`symbol_markup`	optional	free-form	Fancy symbol that can include Greek letters or subscripts and superscripts expressed with the Pango markup language. It is used for presentation in the application so, while it is optional, it is recommended to at least define it identically to `symbol`.
`power_xy`	optional	integer	The power in which the lateral dimensions appear in the quantity. For instance, this is 1 for grain dimensions, 2 for areas and volumes. The default value is 0.
`power_z`	optional	integer	The power in which the "height" dimension appears in the quantity. For instance, this is 1 for values and volumes, 0 for dimensions and areas. The default value is 0.
`same_units`	optional	0 or 1	Given as 1 if the quantity makes sense only for lateral and "height" dimensions being the same physical quantities. For instance, this is required for the surface area. The default is 0.
`is_angle`	optional	0 or 1	Given as 1 if the quantity is an angle. The expression should calculate angles in radians. However, if `is_angle` is set Gwyddion knowns the value can be converted to degrees for presentation. The default is 0.

At present, user-defined grain quantities cannot depend on other user-defined grain quantities to avoid circular dependencies. The built-in grain quantities are listed below:

Symbol	Group	Name
x_c	Position	Center x position
y_c	Position	Center y position
z_min	Value	Minimum value
z_max	Value	Maximum value
z_m	Value	Mean value
z_med	Value	Median value
z_rms	Value	RMS value
b_min	Value	Minimum value on boundary
b_max	Value	Maximum value on boundary
A_px	Area	Pixel area
A_0	Area	Projected area
A_s	Area	Surface area
a_eq	Area	Equivalent square side
r_eq	Area	Equivalent disc radius
A_h	Area	Area above half-height
A_c	Area	Area of convex hull
V_0	Volume	Zero basis volume
V_min	Volume	Grain minimum basis volume
V_L	Volume	Laplacian background basis volume
L_b0	Boundary	Projected boundary length
D_min	Boundary	Minimum bounding size
phi_min	Boundary	Minimum bounding direction
D_max	Boundary	Maximum bounding size
phi_max	Boundary	Maximum bounding direction
R_i	Boundary	Maximum inscribed disc radius
x_i	Boundary	Maximum inscribed disc center x position
y_i	Boundary	Maximum inscribed disc center y position
R_e	Boundary	Minimum circumcircle radius
x_e	Boundary	Minimum circumcircle center x position
y_e	Boundary	Minimum circumcircle center y position
R_m	Boundary	Mean radius
M_min	Boundary	Minimum Martin diameter
omega_min	Boundary	Direction of minimum Martin diameter
M_max	Boundary	Maximum Martin diameter
omega_max	Boundary	Direction of maximum Martin diameter
theta	Slope	Inclination ϑ
phi	Slope	Inclination φ
x_0	Curvature	Curvature center x position
y_0	Curvature	Curvature center y position
z_0	Curvature	Curvature center z value
kappa_1	Curvature	Curvature 1
kappa_2	Curvature	Curvature 2
phi_1	Curvature	Curvature angle 1
phi_2	Curvature	Curvature angle 2
a_e1	Moment	Major semiaxis of equivalent ellipse
a_e2	Moment	Minor semiaxis of equivalent ellipse
phi_e1	Moment	Orientation of equivalent ellipse

For instance, a new grain value Height, measuring the grain height as the difference between the maximum and minimum value, can be defined as follows:

```
Gwyddion resource GwyGrainValue
symbol dz
symbol_markup Δz
power_xy 0
power_z 1
expression z_max - z_min
```

Raw File Presets

Raw file presents reside in directory `rawfile`, they are identified by GwyRawFilePreset and contain only named parameters. They are normally created and edited by the preset editor in the raw file import module.

The named parameters in the resource files correspond closely to the parameters in the user interface explained in detail in section Raw Data File Import. Hence, they will be described only briefly here.

Parameter	Type	Description
`xres`, `yres`	integer	horizontal and vertical size
`xreal`, `yreal`	number	physical dimensions, in units given by `xyexponent` and `xyunit`
`xyexponent`	multiple of 3	power of 10 to multiply `xreal` and `yreal` with
`xyunit`	string	base units of `xreal` and `yreal`, e.g. `"m"`
`zscale`	number	unit step in values
`zexponent`	multiple of 3	power of 10 to multiply `zscale` with
`zunit`	string	base units of `zscale`
`havemissing`	0 or 1	0 means missing value handling is disabled, 1 means it is enabled
`missingvalue`	number	the special subsitute value that denotes missing data
`format`	0 or 1	0 means binary, 1 means text
`builtin` (binary)	integer	built-in data format id, see below
`offset` (binary)	integer	data offset in file, in bytes
`size` (binary)	integer	data value size, in bits
`skip` (binary)	integer	number of bits to skip after each value
`rowskip` (binary)	integer	number of additional bits to skip after each row
`sign` (binary)	0 or 1	0 means unsigned, 1 means signed
`revsample` (binary)	0 or 1	1 means reverse bits in values
`revbyte` (binary)	0 or 1	1 means reverse bits in bytes
`byteswap` (binary)	integer	byte swap pattern
`lineoffset` (text)	integer	lines to skip before starting to read the data
`skipfields` (text)	integer	fields to skip at the start of each line
`delimiter` (text)	string	field delimiter, empty string means arbitrary whitespace
`decomma` (text)	0 or 1	1 if decimal separator is comma, 0 for dot

Note the choice of a built-in binary format, i.e. nonzero `builtin`, implies the binary format to some extent. This means the options `size`, `revbyte` and `sign` are ignored as they are used only for detailed specification of user formats. The available formats are listed in the following table:

Type	Description
0	user-specified
1	signed 8bit integer
2	unsigned 8bit integer
3	signed 16bit integer
4	unsigned 16bit integer
5	signed 32bit integer
6	unsigned 32bit integer
7	IEEE float
8	IEEE double
9	signed 64bit integer
10	unsigned 64bit integer

5.10 Settings

Gwyddion module functions remember parameters values between invocations and also between individual sessions. The place where all the values are stored is called settings. The settings include a few program-wide parameters as well.

The permanent storage for the settings is the file `settings` in a user's directory, this usually means under `~/.gwyddion` (Unix) or `Documents and Settings\gwyddion` (MS Windows). The file is only read when Gwyddion starts and written when it terminates. You should keep this in mind if you want to do some manual modifications. Unknown entries in the settings are ignored but preserved.

The settings file starts with a magic header line

```
Gwyddion Settings 1.0
```

followed by lines with individual parameters and values (that form, technically, a serialised GwyContainer). Gwyddion writes the entries in the alphabetical order but this is not a requirement and you do not have to keep the order when modifying the file.

Each parameter line has the form

```
"key" type value
```

Typical module settings keys start with `/module/`*`modulename`*, although in a few cases the module name part is not actually the module name, either because several modules share settings or for historical reasons. Program-wide setting keys start with `/app/`. All possible value types are listed in the following table.

Type	Description
boolean	Logical value that can be either `True` or `False`.
char	Single character. Normal characters are represented directly using a single character. Special characters are represented using the hexadecimal notation as `0x`*`XX`*. This parameter type is not actually used much by modules.
int32	Signed 32bit integer. Gwyddion writes them in decimal notation but reads also other notations such as hexadecimal.
int64	Signed 64bit integer. Gwyddion writes them in decimal notation but reads also other notations such as hexadecimal.
double	Floating point number. They can be in the scientific format, e.g. `1.23e-4`. They are represented in the standard C/POSIX locale, i.e. decimal dot is used (not comma or other separators).
string	String of characters in double quotes, generally UTF-8 encoded. Special characters, including contained double quotes, are escaped using the standard backslash notation.

Some potentially useful program-wide options that can be modified by editing the settings file:

Key	Type	Description
/app/restore-tool-position	boolean	If set to `True`, Gwyddion restores not only size of tool dialogs but also their positions (if possible). For well-behaved window managers this is more likely to be annoying than helpful but in MS Windows you might want to try enabling it.
/app/3d/axes/disable	boolean	If set to `True`, axis labels will never be drawn on OpenGL 3D views, even if enabled. This can help with certain troublesome 3D driver/card/GtkGLExt combinations in which Gwyddion is likely to crash when it tries to draw the axes.
/app/help/user-guide-base	string	Base location of the user guide for help. If not set, the default on-line location is used, i.e. something like `"https://gwyddion.net/documentation/user-guide-en"`, depending on the language. If you want to use a local copy of the HTML guide, set this setting to directory name, for instance `"/home/yeti/docs/gwyddion-user-guide-xhtml-en-2014-09-17"`.

5.11 Toolbox Configuration

The lower part of the toolbox containing the buttons for functions and tools can be modified from within the program using the toolbox editor *Edit* → *Toolbox*.

It can be also customised by editing file `ui/toolbox.xml`. This is also the file you should copy to another account or machine to replicate the toolbox configuration there. Similarly to custom keyboard shortcuts, the file located in the user's directory, which usually means `~/.gwyddion` (Unix) or `Documents and Settings\gwyddion` (MS Windows). A good starting point for customisation is the default `ui/toolbox.xml` file installed with Gwyddion under `share/gwyddion`.

The number buttons in a row is controlled by the `width` attribute of the top level element `toolbox`. To change it to five just change the begining of the file to

```
<toolbox width='5'>
```

Expandable and collapsable groups of buttons such as *Data Process* or *Tools* are created with tag `group`. You can create as many or as few groups as you want. Functions of different kinds can placed in one group if you wish. Each group must be identified with a unique `id` attribute which is, among other things, used to remember which groups were collapsed and which were expanded. The attribute `title` determines the title:

```
<group id='proc' title='Data Process' translatable='yes'>
```

Groups can optionally have a `translatable` attribute that specifies if the title should be translated. This is most useful for groups present in the default toolbox configuration. When you add custom groups you will probably already give them names in your preferred language.

Individual buttons are created with elements `item`:

```
<item type='proc' function='align_rows' icon='gwy_line_level' run='non-interactive'/>
```

Each item must have the `type` attribute, defining the function type. Unless the type is `'empty'` it must also have a `function` attribute defining the specific function. Function names can be located in the module browser (*Info* → *Module Browser*), where they are listed in *Registered functions* for each module; or in the on-line module browser. The available function types are listed in the following table:

Type name	Function kind
`empty`	Empty placeholder that can be used for separation or row alignment.
`builtin`	A built-in function, which includes zooming in and out and 3D view activation. There are exacly four: `'display_3d'`, `'zoom_in'`, `'zoom_out'` and `'zoom_1_1'`.
`proc`	A two-dimensional data (image) processing function. It has prefix `proc::` in the module browser.
`graph`	A graph function. It has prefix `graph::` in the module browser.
`volume`	A volume data function. It has prefix `volume::` in the module browser.

Type name	Function kind
`xyz`	An XYZ data function. It has prefix `xyz::` in the module browser.
`tool`	A tool. It has prefix `tool::` in the module browser.

Data processing functions (incldluing volume and XYZ) can optionally have a `run` attribute that specifies how the function should be invoked. When it is set to `non-interactive` the function is executed immediately, without any dialogue asking for parameters (similar to how pressing **Ctrl-F** runs the last used image processing function). The opposite is `interactive` which generally does not need to be specified because it is also the default mode. Note that not all functions can be run in both modes: some simple functions never take any user input, other functions always require user input.

The button icon is specified using the `icon` attribute. Some module functions have icons predefined (so you do not have to specify it) but not all have because the number of available functions is huge. A Gwyddion stock icon can be used or possibly a GTK+ stock icon. Note Gwyddion icon names have words separated with underscores while GTK+ icon names use dashes.

If you cannot choose from the provided set of icons it is also possible to draw your own icon and put it to `~/.gwyddion/pixmaps` (or its equivalent on other systems), using the same naming convention as Gwyddion icons. It may be useful to start from the GIMP XCF source images for the icons since they contain individual pieces that can be mixed and matched. If you draw a nice icon you are of course encouraged to submit it for inclusion in Gwyddion.

Since tools are accessible only from the toolbox, not listing a tool in `ui/toolbox.xml` renders it unavailable. Therefore, a special empty item

```
<item type='tool'/>
```

can be used to place all tools that have not been explicitly placed yet to the corresponding position (in a pretty much arbitrary order).

5.12 Format of Gwyddion Files

A Gwyddion native data file (GWY) consists of a tree-like structure of serialized objects. Generally, these objects can be of various kind and contain other embedded objects (hence the tree-like structure). It can be instructive to play with **gwydump**, a simple file structure visualizer available in on the project's web, for a while and examine the contents of various files. If you plan to read and/or write GWY files in independent software, have also a look at libgwyfile, a small standalone embeddable library for GWY file handling. Third party implementations of the file format also exist (with varying capabilities), for instance Python gwyfile.

The Two Layers

This file format description specifies in fact two different things that are sometimes useful to distinguish:

- The physical structuring of the files: byte order, data kinds and their representation, how the sizes of various bits of data are determined, how data objects are nested to form the tree, etc. It is called the *generic GWY file format* in libgwyfile. It can be in principle used for something completely different than SPM data by some completely unrelated software – and would be still called the GWY file format.
- The representation of particular SPM data by Gwyddion, specifying further conventions on top of the physical structure and interpretation. For instance we describe that images are stored as objects called GwyDataField that have resolutions stored this way, image data stored that way, etc. Libgwyfile calls it the *Gwyddion GWY file format*.

In should be usually quite clear where we talk about the generic format and where about specific Gwyddion data objects (occassionally it will be explicitly noted). We will start with the physical file structure.

Byte Order

All data are stored in little-endian (also known as LSB or Intel) byte order.

File Header

The file header consists of four bytes (magic number) with the values of ASCII characters `GWYP`.

This is the new file format; an older version of file format with magic header `GWYO` also exists. It will not be discussed here as it is mostly extinct nowadays.

File Data

The rest of a Gwyddion file consists of a serialized GwyContainer object that contains all the data. This top-level object is stored exactly the same way as any other object, that is as described in the next section.

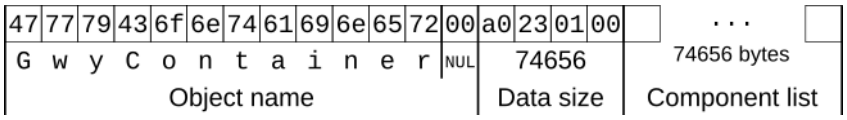
A generic GWY file can have some other object as the top-level object.

Object Layout

An object consists of three parts (in the following order):

1. Type name, stored as a `NUL`-terminated string of ASCII characters. In a Gwyddion file this is the type name in GObject type system. Generally the name should be a valid C identifier.
2. Serialized data size, stored as an unsigned 32bit integer. It does not include the size of the type name and the size of self.
3. Component list. Components are named parts of object data, each of particular data type: an atomic type, an array of atomic types, or again an object. They are stored in no particular order and file readers must not assume any specific component order.

The number of components is not explicitly stored. It is implied by the total serialized data size. The components take together exactly this many bytes; there is no padding.



Object layout illustrated for a `GwyContainer`.

Components

Each component consists of three parts (in the following order):

1. Name, stored as a `NUL`-terminated UTF-8 encoded string.
2. Type, stored as a single unsigned byte (character). The table of possible component types is presented below.
3. Data, stored as whatever is appropriate for a particular type.

Components are often called items in libgwyfile and Gwyddion libraries.

Component size is not explicitly represented – it is implied by its type and contents. Skipping an unknown (or uninteresting) component while reading a GWY file can be done by first reading the component name and type. Then there are several possibilities:

- If the type is atomic and fixed-size the number of bytes to skip is given by the type table.
- For strings, you have to skip after the terminating `NUL` character.
- If the type is an object you need to read its name and size, and the size then tells you exactly how many bytes to skip further.
- If the type is an array of simple atomic types then read the array length and then multiply the atomic type size with the length. This gives the number of bytes to skip.
- For an array of strings, read the array length and repeat string skipping (move past `NUL`) according to the array length.
- Finally, the most complex scenario is an array of objects when you need to read the array length and then repeat the object skipping the specified number of times.

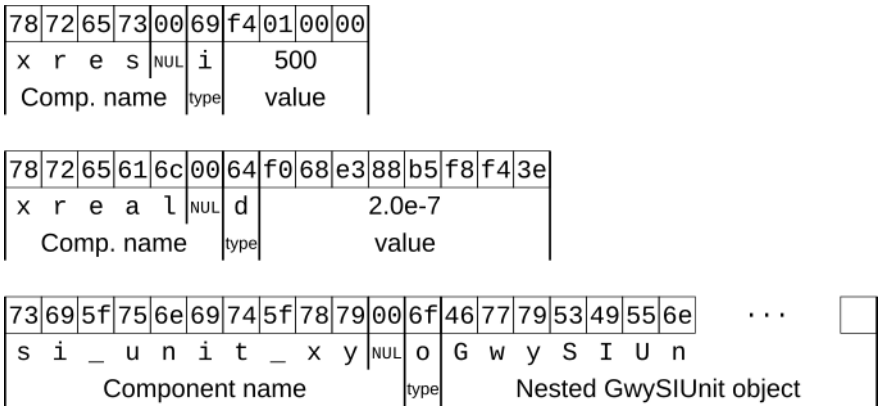


Illustration of representation of three components, integer `xres`*, floating point* `xreal` *and nested object* `si_unit_xy`*. They would simply follow one another in the file.*

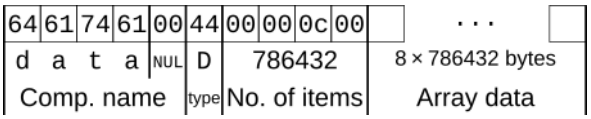
Data Types

Available atomic data types are listed in following table:

Type	Character	Size	Note
boolean	`b`	1 byte	Zero is false, nonzero (normally 1) is true.
character	`c`	1 byte	
32bit integer	`i`	4 bytes	
64bit integer	`q`	8 bytes	
double	`d`	8 bytes	Finite IEEE 754 double precision floating point number, i.e. files must not contain infinities and not-a-numbers.
string	`s`	variable	`NUL`-terminated and UTF-8 encoded.
object	`o`	variable	Nested serialized object as described above.

Each atomic type except boolean has its array counterpart. The type character of array types is the same as of the corresponding atomic type, except it is uppercase. An array is stored as unsigned 32bit integer representing the number of items, followed by the item values. The number of items must be positive; empty arrays are not stored. Array data types are listed in following table:

Type	Character	Note
array of characters	`C`	In general neither `NUL`-terminated nor UTF-8 encoded (`s` is used for text strings).
array of 32bit integers	`I`	
array of 64bit integers	`Q`	
array of doubles	`D`	
array of strings	`S`	
array of objects	`O`	Uppercase Oh, not zero.



Array component layout illustrated for an array of doubles called `data`*.*

Top-Level GwyContainer

GwyContainer is a general dictionary-like data object that can hold arbitrary components of arbitrary types. This permits incorporating unforeseen data structures into GWY files in a relatively sane manner.

The names (keys) of data objects in a GwyContainer representing a Gwyddion GWY file strongly resemble UNIX file names, i.e. they have the form of /-separated paths and form a sort of tree-like structure. For instance the title of the first image, numbered 0, is stored under the key `/0/data/title`. Note that some data or information is found under keys that may not seem logical; the reason is usually historical.

The following sections describe the organisation of interesting data and information in the top-level GwyContainer. The list is not necessarily complete. However, since all data items in the file specify consistently their name, type and size in bytes it is always possible to skip unknown data types or data you are not interested in and extract only the desired data items.

Images

The following table summarises the common keys of image-related data in the top-level container for image number 0. For other images, the number 0 has to be replaced with the corresponding image number. Note that images are often numbered sequentially, starting from 0. However, they can have any numbers and the set of images numbers does not have to be contiguous.

Key	Type	Meaning
`/0/data`	GwyDataField	Channel data.
`/0/data/title`	string	Channel title, as shown in the data browser.
`/0/data/visible`	boolean	Whether the image should be displayed in a window when the file is loaded.
`/0/data/realsquare`	boolean	Whether the image should be displayed as *Physically square* (as opposed to *Pixelwise square*).
`/0/base/palette`	string	Name of the false color gradient used to display the image.
`/0/base/range-type`	32bit integer	False color mapping type (as set by the Color range tool), the value is from GwyLayerBasicRangeType enum.
`/0/base/min`	double	Minimum value for user-set display range.
`/0/base/max`	double	Maximum value for user-set display range.
`/0/mask`	GwyDataField	Mask data. The pixel dimensions of this data field must match those of the image data.
`/0/mask/red`	double	Red component of the mask color.
`/0/mask/green`	double	Green component of the mask color.
`/0/mask/blue`	double	Blue component of the mask color.
`/0/mask/alpha`	double	Alpha (opacity) component of the mask color.
`/0/show`	GwyDataField	Presentation data. The pixel dimensions of this data field must match those of the image data.
`/0/meta`	GwyContainer	Channel metadata. The keys are directly the names as displayed in the metadata browser and the string values are the values.
`/0/data/log`	GwyStringList	Channel log as a list of string log entries. They have the format *type*`::`*function*`(`*param*`=`*value*`, ...)@`*time*.
`/0/select/`*foo*	a GwySelection subclass	Selection data. Each kind of selection has (usually) a different object type and is stored under a different name; the specific name *foo* is the same as shown in the selection manager.

Channels are represented as GwyDataField objects. The components of a GwyDataField are summarised in the following table:

Component	Type	Meaning
`xres`	32bit integer	Horizontal size in pixels.
`yres`	32bit integer	Vertical size in pixels.
`xreal`	double	Horizontal dimension in physical units.
`yreal`	double	Vertical dimension in physical units.

Component	Type	Meaning
xoff	double	Horizontal offset of the top-left corner in physical units. It usually occurs only if non-zero.
yoff	double	Vertical offset of the top-left corner in physical units. It usually occurs only if non-zero.
si_unit_xy	GwySIUnit	Unit of lateral dimensions.
si_unit_z	GwySIUnit	Unit of data values.
data	array of doubles	Field data, stored as a flat array of size xres × yres, from top to bottom and from left to right.

Graphs

The following table summarises the common keys of graph-related data in the top-level container for graph number 1. For other graphs, the number 1 has to be replaced with the corresponding graph number. Note that graphs are often numbered sequentially, starting from 1, however, they can have any numbers positive and the set of graph numbers does not have to be contiguous. The number 0 in the prefix of graph keys is a historical relic that does not mean anything and it is always 0.

Key	Type	Meaning
/0/graph/graph/1	GwyGraphModel	Graph model object data.
/0/graph/graph/1/visible	boolean	Whether the graph should be displayed in a window when the file is loaded.

Graphs are represented as GwyGraphModel objects. The components of a GwyGraphModel are summarised in the following table:

Component	Type	Meaning
curves	array of GwyGraphCurveModels	Individual graph curves.
title	string	Graph title as displayed in the data browser.
x_unit	GwySIUnit	Unit of the abscissa.
y_unit	GwySIUnit	Unit of the ordinate.
top_label	string	Label on the top axis.
bottom_label	string	Label on the bottom axis.
left_label	string	Label on the left axis.
right_label	string	Label on the right axis.
x_is_logarithmic	boolean	Whether the abscissa has a logarithmic scale.
y_is_logarithmic	boolean	Whether the ordinate has a logarithmic scale.
x_min	double	User-set minimum value of the abscissa.
x_min_set	boolean	Whether user-set minimum value of the abscissa should be used (otherwise the range is determined automatically).
x_max	double	User-set maximum value of the abscissa.
x_max_set	boolean	Whether user-set maximum value of the abscissa should be used (otherwise the range is determined automatically).
y_min	double	User-set minimum value of the ordinate.
y_min_set	boolean	Whether user-set minimum value of the ordinate should be used (otherwise the range is determined automatically).
y_max	double	User-set maximum value of the ordinate.

Component	Type	Meaning
y_max_set	boolean	Whether user-set maximum value of the ordinate should be used (otherwise the range is determined automatically).
grid-type	32bit integer	Type of grid shown. The value is from GwyGraphGridType enum.
label.has_frame	boolean	Whether the graph key has a frame.
label.frame_thickness	32bit integer	Width of graph key frame.
label.reverse	boolean	Whether to reverse the graph key.
label.visible	boolean	Whether the graph key is visible.
label.position	32bit integer	The position (corner) where the graph key is places. The value is from GwyGraphLabelPosition enum.

Graph curves are represented as GwyGraphCurveModel objects. The components of a GwyGraphCurveModel are summarised in the following table:

Component	Type	Meaning
xdata	array of doubles	Abscissa points. The number of points must match ydata.
ydata	array of doubles	Ordinate points. The number of points must match xdata.
description	string	Curve description (name).
type	32bit integer	Curve mode (points, lines, etc.) The value is from GwyGraphCurveType enum.
color.red	double	Red component of the curve color.
color.green	double	Green component of the curve color.
color.blue	double	Blue component of the curve color.
point_type	32bit integer	Type of symbols representing data points. The value is from GwyGraphPointType enum.
point_size	32bit integer	Size of symbols representing data points.
line_type	32bit integer	Type of lines connecting data points. The value is from GwyGraphLineType enum.
line_size	32bit integer	Width of lines connecting data points.

Spectra

The following table summarises the common keys of spectra-related data in the top-level container for spectra set number 0. For other spectra, the number 0 has to be replaced with the corresponding spectra set number. Note that spectra sets are often numbered sequentially, starting from 0, however, they can have any numbers and the set of spectra set numbers does not have to be contiguous.

	Key	Type	Meaning
	/sps/0	GwySpectra	Spectra data.

Sets of spectra of one kind are represented as GwySpectra objects. The components of a GwySpectra are summarised in the following table:

Component	Type	Meaning
title	string	Spectra title as displayed in the data browser.
si_unit_xy	GwySIUnit	Unit of spectrum position coordinates.

Component	Type	Meaning
coords	array of doubles	Coordinates of points where the spectra were taken, in physical units. Each spectrum takes two items: for the horizontal and vertical coordinate. The number of coordinates must match the number of curves in data.
data	array of GwyDataLines	Individual spectra curves.
selected	array of 32bit integers	Indices of selected spectra curves.

Individual curves in spectra are represented as GwyDataLine objects. The GwyDataLine object is a one-dimensional counterpart of GwyDataField and is used also for other regular one-dimensional data. The components of a GwyDataLine are summarised in the following table:

Component	Type	Meaning
res	32bit integer	Number of data points.
real	double	Length in physical units.
off	double	Offset of the begining in physical units. It usually occurs only if non-zero.
si_unit_x	GwySIUnit	Unit of abscissa.
si_unit_y	GwySIUnit	Unit of data values.
data	array of doubles	Line data, stored as an array of res, from left to right.

Volume data

The following table summarises the common keys of volume-related data in the top-level container for volume data number 0. For other volume data, the number 0 has to be replaced with the corresponding volume data number. Note that volume data are often numbered sequentially, starting from 0, however, they can have any numbers and the set of volume data numbers does not have to be contiguous.

Key	Type	Meaning
/brick/0	GwyBrick	Volume data.
/brick/0/preview	GwyDataField	Two-dimensional data shown when the volume data are displayed in a window.
/brick/0/title	string	Volume data title, as shown in the data browser.
/brick/0/visible	boolean	Whether the volume data should be displayed in a window when the file is loaded.
/brick/0/preview/palette	string	Name of the false color gradient used to display the preview data.
/brick/0/meta	GwyContainer	Volume data metadata. The keys are directly the names as displayed in the metadata browser and the string values are the values.
/brick/0/log	GwyStringList	Volume data log as a list of string log entries. They have the format *type*::*function*(*param*=*value*, ...)@*time*.

Volume data are represented as GwyBrick objects. The components of a GwyBrick are summarised in the following table:

Component	Type	Meaning
xres	32bit integer	Horizontal size in pixels.
yres	32bit integer	Vertical size in pixels.
zres	32bit integer	Depth (number of levels) in pixels.
xreal	double	Horizontal dimension in physical units.
yreal	double	Vertical dimension in physical units.

Component	Type	Meaning
zreal	double	Depthwise dimension in physical units.
xoff	double	Horizontal offset of the top-left corner in physical units. It usually occurs only if non-zero.
yoff	double	Vertical offset of the top-left corner in physical units. It usually occurs only if non-zero.
zoff	double	Depthwise offset of the top-left corner in physical units. It usually occurs only if non-zero.
si_unit_x	GwySIUnit	Unit of horizontal lateral dimensions.
si_unit_y	GwySIUnit	Unit of vertical lateral dimensions.
si_unit_z	GwySIUnit	Unit of depthwise dimensions.
si_unit_w	GwySIUnit	Unit of data values.
data	array of doubles	Field data, stored as a flat array of size xres × yres × zres, from the zeroth to the last plane, top to bottom and from left to right.
calibration	GwyDataLine	Calibration of the z axis to represent non-linear sampling in this dimension. The number of points must be equal to zres. This component is present only if non-linear sampling is used.

XYZ data

The following table summarises the common keys of XYZ-related data in the top-level container for XYZ data number 0. For other XYZ data, the number 0 has to be replaced with the corresponding XYZ data number. Note that XYZ data are often numbered sequentially, starting from 0, however, they can have any numbers and the set of XYZ data numbers does not have to be contiguous.

Key	Type	Meaning
/xyz/0	GwySurface	XYZ data.
/xyz/0/preview	GwyDataField	Regularised preview shown when the XYZ data are displayed in a window. Note that although XYZ previews are stored in GWY files, they are commonly re-created and updated when the data are displayed so it is rarely useful to add them when you are writing a GWY file.
/xyz/0/title	string	XYZ data title, as shown in the data browser.
/xyz/0/visible	boolean	Whether the XYZ data should be displayed in a window when the file is loaded.
/xyz/0/preview/palette	string	Name of the false color gradient used to display the preview data.
/xyz/0/meta	GwyContainer	XYZ data metadata. The keys are directly the names as displayed in the metadata browser and the string values are the values.
/xyz/0/log	GwyStringList	Volume data log as a list of string log entries. They have the format *type*::*function*(*param*=*value*, ...)@*time*.

XYZ data are represented as GwySurface objects. The components of a GwySurface are summarised in the following table:

Component	Type	Meaning
si_unit_xy	GwySIUnit	Unit of horizontal lateral dimensions.
si_unit_z	GwySIUnit	Unit of data values.
data	array of doubles	XYZ data, stored as a flat array of whose size is a multiple of 3. Each XYZ triplet is stored together, leading to the following data order: x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, etc.

Curve map data

The following table summarises the common keys of curve map related data in the top-level container for XYZ data number 0. For other curve map data, the number 0 has to be replaced with the corresponding curve map data number. Note that curve map data are often numbered sequentially, starting from 0, however, they can have any numbers and the set of curve map data numbers does not have to be contiguous.

Key	Type	Meaning
`/lawn/0`	GwyLawn	Curve map data.
`/lawn/0/preview`	GwyDataField	Image preview shown when the curve map data are displayed in a window.
`/lawn/0/title`	string	XYZ data title, as shown in the data browser.
`/lawn/0/visible`	boolean	Whether the XYZ data should be displayed in a window when the file is loaded.
`/lawn/0/preview/palette`	string	Name of the false color gradient used to display the preview data.
`/lawn/0/preview/realsquare`	boolean	Whether the image should be displayed as *Physically square* (as opposed to *Pixelwise square*).
`/lawn/0/meta`	GwyContainer	Curve map data metadata. The keys are directly the names as displayed in the metadata browser and the string values are the values.
`/lawn/0/log`	GwyStringList	Volume data log as a list of string log entries. They have the format *`type`*`::`*`function`*`(`*`param=value`*`, ...)@`*`time`*.

Curve map data are represented as GwyLawn objects. The components of a GwyLawn are summarised in the following table:

Component	Type	Meaning
`xres`	32bit integer	Horizontal size in pixels.
`yres`	32bit integer	Vertical size in pixels.
`ncurves`	32bit integer	The number of curves in each pixel.
`curvelengths`	array of 32bit integers	Array of size `xres` × `yres` containing the sample counts in each pixel (in the usual image order). The sample counts can be zero.
`xreal`	double	Horizontal dimension in physical units.
`yreal`	double	Vertical dimension in physical units.
`xoff`	double	Horizontal offset of the top-left corner in physical units. It usually occurs only if non-zero.
`yoff`	double	Vertical offset of the top-left corner in physical units. It usually occurs only if non-zero.
`si_unit_xy`	GwySIUnit	Unit of horizontal lateral dimensions.
`si_units_curves`	array of GwySIUnit objects	Units of curve data, as an array with `ncurves` items.
`data`	array of doubles	Concatenated curve data, stored as a flat array of whose size is a multiple of the number of curves (`ncurves`). The data are stored in the usual image order. First all curves in the top left pixel are concatenated and stored, then the all curves in the next pixel, etc. Reading and writing this item requires the use of `curvelengths` for correct indexing.
`curve_labels`	array of strings	Optional array with `ncurves` items specifying curve labels.
`nsegments`	32bit integer	The number of segments each curve is split into – if curves are segmented. In such case item `segments` should specify the segmentation.

Component	Type	Meaning
segments	array of 32bit integers	Segmentation of each curve, given as start and end indices of each segment in each curve. The array size is xres × yres × 2nsegments. Segments do not have to exactly partition the curve; they can overlap or leave holes.
segment_labels	array of strings	Optional array with nsegments items specifying segment labels.

Other Items

The main GwyContainer contains also items that do not pertain to any specific channel, graph or other data type.

Key	Type	Meaning
/filename	string	The name of file the GwyContainer is currently associated with. If it was saved to a file then the item contains the corresponding file name. Otherwise it contains the name of file from which the data were loaded or imported. It may also be unset, for instance for newly created synthetic data.

Auxiliary Objects

The components of a GwySIUnit are summarised in the following table:

Component	Type	Meaning
unitstr	string	Textual representation of the unit, e.g. "A" or "m^-1" (as base SI unit, prefixes are ignored).

The components of a GwySelection are summarised in the following table. Some selection types can have other data members; refer to the documentation of specific selection classes for how to interpret the data.

Component	Type	Meaning
max	32bit integer	Maximum number of objects the selection can hold (this is the number set by gwy_selection_set_max_objects()).
data	array of doubles	Selection data. The number of items that form one selection object is determined by the selection type.

The components of a GwyStringList are summarised in the following table. Note that if GwyStringLists are used to represent logs, the strings have a specific structure described above.

Component	Type	Meaning
strings	array of strings	List of string items.

5.13 Simple Field Files

The Gwyddion native file format captures all the information and state Gwyddion needs to save and consequently it is quite complex. Often it is not practical to save files in .gwy format in custom programs and scripts creating input for Gwyddion.

The Gwyddion Simple Field file format (.gsf) can be used in these situations instead. It is a single-channel format for 2D data that was designed to be easy and efficient to read and write, with human-readable header, reasonably expressive, and avoiding instrument or application specific fields (though it can optionally bear them).

GSF can be read and written by Gwyddion version 2.20 or later. Third party implementations also exist, for instance Python gsffile.

Overall structure

A GSF file consists of four parts, in the following order:

magic line Files begin with a "magic line" identifying the file type.

text header The header consists of lines of the form

```
name = value
```

defining individual parameters.

NUL padding The header is terminated by one to four NUL bytes, aligning the data start to a multiple of 4.

binary data Binary data is in 32bit floating-point format.

Magic line

GSF files start with the line

```
Gwyddion Simple Field 1.0
```

terminated by a linefeed character (`\n`, ASCII 0x0a).

Text header

Each header line has the form

```
name = value
```

where any whitespace before the name, around the equal sign and at the end of value is ignored. Field names are case-sensitive and follow the usual rules for identifiers in programming languages.

Similarly to the magic line, the lines in the text header are terminated by a linefeed character as is usual on Unix. This means the header must be read and written in binary mode to ensure preservation of end-of-line characters on other systems (and not changing the header size e.g. by LF → CRLF transformation).

Any non-ASCII characters, that can occur for example in the channel title, are represented in UTF-8 encoding. The NUL character may not occur in the header.

Header fields:

Name	Type	Value
`XRes`	Mandatory	The horizontal size in pixels, a positive integer.
`YRes`	Mandatory	The vertical size in pixels, a positive integer.
`XReal`	Optional	Horizontal size in physical units (given by `XYUnits`), a positive floating point number. It defaults to 1.0 if not given.
`YReal`	Optional	Vertical size in physical units (given by `XYUnits`), a positive floating point number. It defaults to 1.0 if not given.
`XOffset`	Optional	Horizontal offset in physical units (given by `XYUnits`), a floating point number. It defaults to 0.0 if not given.
`YOffset`	Optional	Vertical offset in physical units (given by `XYUnits`), a floating point number. It defaults to 0.0 if not given.
`Title`	Optional	Data/channel title. It has no default, applications might display 'Unknown' or something similar if not given.
`XYUnits`	Optional	Lateral units, i.e. units of physical sizes and offsets. They must be given as base units, that is `m` or `A` with no power-of-10 prefix (Gwyddion could deal with it but it might present a problem for other software). The default is no units. This means in SPM data, you normally wish to specify `XYUnits` as `m` because the lateral dimensions are in metres.
`ZUnits`	Optional	Value units, i.e. units of data values. See `XYUnits` above for details.

Floating point numbers can be in the scientific format, e.g. `1.23e-4`. They are represented in the standard C/POSIX locale, i.e. decimal dot is used (not comma or other separators).

The header may contain other fields beside those listed above. Gwyddion will load them into metadata. Common informational fields can include `Comment`, `Date` or `Direction`.

Fields may occur in any order, nevertheless, it is recommended to start with mandatory fields, continue with optional fields and put custom fields last.

A simple header example (also including the magic line):

```
Gwyddion Simple Field 1.0
XRes = 400
YRes = 400
XReal = 5e-05
YReal = 5e-05
XYUnits = m
ZUnits = V
Title = ADC2
```

NUL padding

The text header is followed by one to four NUL (`\0`, ASCII 0x00) bytes that (a) terminate it and (b) align the data start to an offset from the begining of file that is a multiple of 4. More precisely, denoting N the total length of the magic line and the text header, the data starts at the nearest multiple of 4 larger than N.

This padding to a multiple of 4 ensures aligned memory access when mapping the file directly to memory. The number of NUL bytes is uniquely determined by the remainder of the length modulo four (N mod 4):

Remainder	Number of padding NULs
0	4
1	3
2	2
3	1

Binary data

Data values are stored as IEEE 32bit single-precision floating point numbers, in little-endian (LSB, or Intel) byte order. Values are stored by row, from top to bottom, and in each row from left to right.

Similarly to GWY files, the data should not contain infinities and NaNs (not-a-numbers). Gwyddion replaces and masks such values upon import since version 2.57, but behaviour in older versions is undefined.

The physical units of these values are `ZUnits`.

The size of the image data is exactly `4*XRes*YRes` bytes and there is no data after it in the file.

5.14 Simple XYZ Files

Although Gwyddion does work with general XYZ data and raw XYZ data are interpolated to a grid upon import, need has arisen for a file format similar in spirit to Gwyddion simple field (.gsf) but representing the data in XYZ format. Such file format, called Gwyddion XYZ Field (`.gxyzf`), is described in this section.

It should be noted that Z simply stands for the ordinate here. Z values in the file may be actual Z coordinates (heights) but they may also be currents, voltages, etc.

GXYZF can be written by Gwyddion version 2.31 or later. They can also be read since this version, although the regularisation to a grid may be somewhat crude.

Overall structure

A GXYZF file consists of four parts, in the following order:

magic line Files begin with a "magic line" identifying the file type.

text header The header consists of lines of the form

```
name = value
```

defining individual parameters.

NUL padding The header is terminated by one to eight NUL bytes, aligning the data start to a multiple of 8.

binary data Binary data is in 64bit floating-point format.

Magic line

gxyzf files start with the line

```
Gwyddion XYZ Field 1.0
```

terminated by a linefeed character (`\n`, ASCII 0x0a).

Text header

Each header line has the form

```
name = value
```

where any whitespace before the name, around the equal sign and at the end of value is ignored. Field names are case-sensitive and follow the usual rules for identifiers in programming languages.

Similarly to the magic line, the lines in the text header are terminated by a linefeed character as is usual on Unix. This means the header must be read and written in binary mode to ensure preservation of end-of-line characters on other systems (and not changing the header size e.g. by LF → CRLF transformation).

Any non-ASCII characters, that can occur for example in channel titles, are represented in UTF-8 encoding. The NUL character may not occur in the header.

Header fields:

Name	Type	Value
`NChannels`	Mandatory	Number of value (Z) channels, a positive integer. The values stored for each point include also coordinates X and Y but they are not counted into `NChannels`.
`NPoints`	Mandatory	Number data points in the file.
`XYUnits`	Optional	Lateral units, i.e. units of X and Y values. They must be given as base units, that is `m` or `A` with no power-of-10 prefix (Gwyddion could deal with it but it might present a problem for other software). The default is no units. This means in SPM data, you normally wish to specify `XYUnits` as `m` because the lateral dimensions are in metres.
`ZUnits1`, `ZUnits2`, ...	Optional	Value units, i.e. units of data values for individual channels. Channels are numbered from 1 to `NChannels`. See `XYUnits` above for details.
`Title1`, `Title2`, ...	Optional	Titles of individual channels. Channels are numbered from 1 to `NChannels`. Titles have no default, applications might display 'Unknown' or something similar if not given.
`XRes`	Optional	Hint specifying the preferred horizontal size in pixels if the data are regularised to a grid, a positive integer. Readers are not required to honour it and may interpolate data to grids of different dimensions.
`YRes`	Optional	Hint specifying the preferred vertical size in pixels if the data are regularised to a grid, a positive integer. Readers are not required to honour it and may interpolate data to grids of different dimensions.

The header may contain other fields beside those listed above. Gwyddion will load them into metadata. Common informational fields can include `Comment`, `Date` or `Direction`.

Fields may occur in any order, nevertheless, it is recommended to start with mandatory fields, continue with optional fields and put custom fields last.

A simple header example of a two-channel file (also including the magic line):

```
Gwyddion XYZ Field 1.0
NChannels = 2
NPoints = 457884
XYUnits = m
ZUnits1 = m
ZUnits2 = V
Title1 = Height
Title2 = ADC2
```

NUL padding

The text header is followed by one to eight NUL (`\0`, ASCII 0x00) bytes that (a) terminate it and (b) align the data start to an offset from the begining of file that is a multiple of 8. More precisely, denoting *N* the total length of the magic line and the text header, the data starts at the nearest multiple of 8 larger than *N*.

This padding to a multiple of 8 ensures aligned memory access when mapping the file directly to memory. The number of NUL bytes is uniquely determined by the remainder of the length modulo eight (*N* mod 8):

Remainder	Number of padding NULs
0	8
1	7
2	6
3	5
4	4
5	3
6	2
7	1

Binary data

Data values are stored as IEEE 64bit double-precision floating point numbers, in little-endian (LSB, or Intel) byte order. Points are stored in arbitrary order. Each point is stored as a block of `NChannels+2` values: X, Y and then all ordinate values, in the channel order.

The physical units of the values are given by `XYUnits` for X and Y and then `ZUnits1`, `ZUnits2`, ... for the ordinate values.

The size of the data is exactly `8*NPoints*(NChannels+2)` bytes and there is no data after it in the file.

Chapter 6

Building from Source Code and Development

6.1 Build Dependencies

The following table lists packages required to build Gwyddion from source code. If your operating system has separate development packages for libraries you need them too. The table does not include common software compilation prerequisites like the C compiler or the **make** utility. Operating system specifics are described in following sections dedicated to building on particular operating systems.

Minimum required versions are listed for some packages. If no specific version is listed, the minimum version is so old that it did not seem useful to determine it exactly. Specific environments may be listed in the Dependency column, meaning the package is useful only in this environment.

Package	Version	Dependency	Required for, Notes
pkg-config	0.16	**Required**	Tracking the locations and compiler and linker flags of the various packages.
GTK+ 2	2.24.0	**Required**	Gwyddion user interface. This entry implies the dependencies of GTK+ itself, such as GLib, Gdk-Pixbuf, Pango or Cairo.
GLib	2.32.0	**Required**	Everything. GLib is a base library also required by GTK+, but Gwyddion needs a slightly newer version than strictly required by GTK+.
Pango	1.10	**Required**	All text rendering. This entry implies pangocairo, which is an optional component of Pango and in principle can be disabled. However, it is normally included in Pango packages.
Cairo	1.2	**Required**	All drawing within GTK+. Version at least 1.6 is recommended.
FFTW3	3.1	**Required**	Integral transforms, power spectrum, convolution, deconvolution and correlation operations. If OpenMP is enabled and FFTW3 with OpenMP support is found Gwyddion will be able to utilise multithreaded FFT.
GtkGLExt	1.0	Recommended	OpenGL 3D data views. This entry implies the dependencies of GtkGLExt itself, such as the platform OpenGL libraries and headers.
OpenMP	3.1	Recommended	Speedup of various computation using OpenMP multithread parallelisation. OpenMP is a standard, not specific software, but an additional package may be needed to enable it in your C compiler.
HDF5	1.8.13	Recommended	Import of Asylum Research Ergo, Shilps Lucent, Zygo DATX, FusionScope, EPFL, GXSM4, Matlab 7 MAT and generic NSID and HDF5 data files.
zlib		Recommended	Import of SPML data files and import of gzip-compressed data from other file formats (Createc, NRRD, Surf, Matlab 5, RHK SM4 PRM metadata).
libzip	0.11	Recommended	Import of APE DAX, NanoObserver, NanoScanTech, OpenGPS, ATC SPMxFormat, Sensofar PLUX, Olympus POIR, Keyence VK6 and VK7, ZON, numpy NPZ, Matlab 5 MAT and JPK force data files which use ZIP compression. Alternatively, minizip can be used. Only one of the ZIP libraries is needed.
minizip		Optional	Import of APE DAX, NanoObserver, NanoScanTech, OpenGPS, ATC SPMxFormat, Sensofar PLUX, Olympus POIR, Keyence VK6 and VK7, ZON, numpy NPZ, Matlab 5 MAT and JPK force data files which use ZIP compression. This is an alternative to libzip. Mnizip versions change often; versions 1, 2 and 3 should be OK. Only one of the ZIP libraries is needed.

Package	Version	Dependency	Required for, Notes
zziplib		Optional	Import of APE DAX, NanoObserver, NanoScanTech, OpenGPS, ATC SPMxFormat, Sensofar PLUX, Olympus POIR, Keyence VK6 and VK7, ZON, numpy NPZ, Matlab 5 MAT and JPK force data files which use ZIP compression. This is a fallback alternative to libzip and minizip which are preferred. Only one of the ZIP libraries is needed.
libunique	1.0	Optional	Remote control based on D-BUS or whatever technology is currently in.
Python	2.7	Optional	Pygwy, the Gwyddion Python scripting interface. Not just the interpreter is required to build pygwy, but also the Python headers and development libraries.
PyGTK2	2.10	Optional	Pygwy, the Gwyddion Python scripting interface. You need PyGTK2 including the compile-time parts, i.e. codegen, to build pygwy.
GtkSourceView 2		Optional	Syntax highlighting in the Python scripting console.
bzip2		Optional	Import of bzip2-compressed data from NRRD.
LibXML2		Optional	Import of SPML, APE DAX and Anasys XML data files.
JANSSON		Optional	Import of Park PS-PPT data files.
libpng		Optional	Export of height fields to 16bit greyscale PNG images and import from 16bit PNG images. For common 8bit images, you just need PNG support in Gdk-Pixbuf.
libwebp		Optional	WebP format support for image export.
jpegxl		Optional	JPEG-XL format support for image export.
WebP pixbuf loader		Optional	WebP format support for image import. This is a run-time dependency. It does not matter if you have the pixbuf loader during Gwyddion compilation.
HEIF pixbuf loader		Optional	High Efficiency Image File Format support for image import. This is a run-time dependency. It does not matter if you have the pixbuf loader during Gwyddion compilation.
JPEG-XL pixbuf loader		Optional	JPEG-XL format support for image import. This is a run-time dependency. It does not matter if you have the pixbuf loader during Gwyddion compilation.
OpenEXR		Optional	Import and export of OpenEXR HDR images.
C++ compiler		Optional	Import and export of OpenEXR HDR images and import of other high-depth images.
cfitsio		Optional	Import of Flexible Image Transport System (FITS) files.
desktop-file-utils		Optional, Unix	Basic desktop integration to Freedesktop-conforming environments, such as file associations and installation of Gwyddion to the desktop environments menus.
gtk-mac-integration		Optional, OS X	OS X platform integration such as the global menu.
libXmu		Obsolete, X11	Remote control on X11. This is a standard X Window System library and everyone having X probably has its runtime files. However, since the modularisation of X in Xorg 7.0, it is distributed separately and therefore you might not have its development files installed.

6.2 Compilation on Linux/Unix

Gwyddion Unix build system is based on GNU autotools (autoconf, automake, libtool), like most of current Unix Free and Open Source Software. If you have ever compiled software from source code, you very likely met autotools and already know how to proceed. This section shall describe the compilation procedure in enough detail even for the uninitiated though. File `INSTALL` in the top-level directory of the source tarball contains generic GNU autotools installation instructions.

Quick Instructions

If you know the drill:

```
tar -jxvf gwyddion-2.49.tar.xz
cd gwyddion-2.49
./configure
make
make install
```

Source Unpacking

Unpack the source code tarball with

```
tar -Jxvf gwyddion-2.49.tar.xz
```

replacing 2.49 with the actual version number. It will create directory `gwyddion-2.49` (again, with the actual version number in place of 2.49), **cd** to this directory. All other compilation actions will take place there.

If your operating system does not come with xz you might want to download `gwyddion-2.49.tar.gz` (compressed with gzip) instead and unpack it with

```
tar -zxvf gwyddion-2.49.tar.gz
```

However, modern Unix and Unix-like systems come with both xz and gzip, so the considerably smaller `gwyddion-2.49.tar.xz` should be normally the better choice.

Configuration

Run

```
./configure
```

to configure Gwyddion.

The **configure** shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile` in each directory of the package, a couple of header `.h` files containing system-dependent definitions and a few other system-dependent auxiliary files. Finally, it creates a shell script **config.status** that you can run in the future to recreate the current configuration, and a file `config.log`. This file contains the details of the detection process and it is helpful to include it in compilation related bug reports. At the end, **configure** also prints a summary of enabled/disabled optional features, including the reasons why features were disabled.

If **configure** reports missing required packages, install these packages and re-run it. The same applies to the case when **configure** passes but you find you have not installed an optional package you want to compile Gwyddion with. It is possible a package is not found or it is misdetected even if you have installed it, namely when it is installed into a non-standard directory. In this case it is necessary to adjust certain environment variables to make **configure** able to find the packages:

PKG_CONFIG_PATH Most packages come with so called pkg-config files (`.pc`) that describe how programs should compile and link with them. **configure** uses information from these files, therefore `PKG_CONFIG_PATH` must be set to list all non-standard directories with relevant pkg-config files. To add for instance a GTK+ installation in `/opt/gnome` and a FFTW3 installation in `$HOME/opt/fftw3` one can do

```
PKG_CONFIG_PATH=/opt/gnome/lib/pkgconfig:$HOME/opt/fftw3/lib/pkgconfig
export PKG_CONFIG_PATH
```

PATH, LD_LIBRARY_PATH, DYLD_LIBRARY_PATH It may be necessary to adjust these variables to include non-standard directories with executables and libraries of relevant packages, respectively. Variables `LD_LIBRARY_PATH` and `DYLD_LIBRARY_PA` both set the search path for shared libraries, but the former is used in Linux and BSD systems, while the latter on OS X.

CPPFLAGS, LDFLAGS It may be necessary to adjust these variables to include non-standard directories with header files and libraries of packages that do not come with pkg-config files, for example for libTIFF in `/usr/local` one can set:

```
CPPFLAGS=-I/usr/local/include
export CPPFLAGS
LDFLAGS=-L/usr/local/lib
export LDFLAGS
```

Option `--prefix` of **configure** sets the base installation directory. Program components will be installed into its `bin`, `lib`, `share`, etc. subdirectories (that will be created if they do not exist). More detailed control is possible with options specifying particular subdirectories as `--bindir`, `--libdir`. The default prefix is `/usr/local/bin`, to install Gwyddion into your home directory you may want to use for instance

```
./configure --prefix=$HOME/opt/gwyddion
```

If you install Gwyddion for personal use it is recommended to use a similar installation directory as no steps need to be performed as root in this case.

Configuration tweaks

Optional features can be enabled/disabled with options such as `--with-foo`/`--without-foo` or `--enable-foo`/`--disable-fo` For instance compilation with zlib can be disabled with:

```
./configure --without-zlib
```

By default all optional features are enabled if their prerequisites are found. A brief summary of enabled and disabled optional features is printed near the end of **configure** output.

The complete list of **configure** options and important variables can be obtained with:

```
./configure --help
```

The list is long and most of the options and variables control inclusion/exclusion of individual optional features or provide compiler and linker flags for the various libraries. For instance, by setting `FFTW3_CFLAGS` and `FFTW3_LIBS` you can specify (or override) how compilation and linking with FFTW3 should be done. However, this manual specification is only a fallback if the much more convenient pkg-config method does not work.

Some interesting general options are explained in the following paragraphs.

User's tweaks

Gwyddion comes with various desktop integration files defining MIME types, menu entries, file associations, thumbnailers, etc. If you install Gwyddion to a system prefix they usually end up in the correct location. However, if you install it somewhere to your home directory then these files need to be placed elsewhere, namely into certain dot-directories in your home.

This can be requested using `--enable-home-installation` option of **configure**. Note that using this option causes installation of files outside the specified prefix.

Packager's tweaks

If Gwyddion is installed into a staging area for a subsequent packaging it is necessary to disable certain post-installation actions that need to be done on the target system, not while packaging.

Updating of Freedesktop files can be disabled with `--disable-desktop-file-update`. Installation of GConf2 schemas can be disabled with `--disable-schemas-install`. Usually, this does not have to be done explicitly as installations into a staging area use non-empty `DESTDIR` (see installation). If `DESTDIR` is found to be non-empty the build system skips post-installation actions automatically.

Pasing `--enable-library-bloat` to **configure** enforces linking modules with all libraries. This is automatically enabled on MS Windows where it is required. On Unix systems, linking modules with libraries that are loaded into the main program anyway only needlessly slows things down (both during build and program execution). So modules are not explicitly linked with base libraries such as GLib. If your environment or policy demands linking modules with all libraries using this option will ensure it.

Pasing `--disable-module-bundling` to **configure** prevents the bundling of all modules of the same class (file, process, ...) to a single shared library, which is normally done to save space and speed up program startup. Although bundling changes nothing functionally, it changes the set of installed files considerably. If you, for whatever reason, depend on the file `gwyfile.so` existing on disk then please stop. But meanwhile you can use this option to force the traditional installation where each module was in a separate file.

Developer's tweaks

If you intend to patch or otherwise modify Gwyddion source code pass option `--enable-maintainer-mode` to **configure** to enable various update and rebuild rules that are not used in plain compilation. Depending on the nature of the modifications, some of the additional tools described in section Subversion Checkout, Development may be necessary.

By default, the C API reference documentation is not rebuilt. Pre-built HTML is distributed in the tarball, the documentation seldom changes and generating it takes a rather long time. To enable the generation of API documentation pass option `--enable-gtk-doc` to **configure**. Of course, you also need gtk-doc. Note that **configure** warns if you enable the maintainer mode but disable gtk-doc (which can be useful to avoid pointless repeated documentation rebuilds). Unless you are going to do **`make dist`**, this is harmless.

Compilation

Run

```
make
```

and wait until Gwyddion is compiled. If **configure** finished without errors the compilation should pass too. To shorten the wait, enable parallel compilation by running **make** as

```
make -j6
```

replacing 6 with the number of processor cores.

If you find you need to do unusual things to compile the package, please try to figure out how **configure** could detect whether and what to do, and e-mail patches or instructions to the bug-report address so they can be considered for the next release.

Installation

Gwyddion has to be installed to be run, it is not possible to run it uninstalled.

Run

```
make install
```

to install Gwyddion to the target directory. If you install Gwyddion to a system directory you have to become root for running this command. This is the *only* command that you might have to run as root during the installation. For example using **sudo**:

```
sudo make install
```

To install Gwyddion to a staging area, for example for packaging, set **make** `DESTDIR` variable to a prefix that will be prepended to all target directories:

```
make install DESTDIR=/var/tmp/gwyddion-buildroot
```

Do *not* override individual directory variables as `bindir`, `libdir`.

If you do not install to a system directory, e.g. to a subdirectory of your home directory, you may need to adjust the following variables during installation:

- `GCONF_SCHEMA_CONFIG_SOURCE` – location of GConf2 schemas
- `KDE4_MODULE_DIR` – location of KDE4 modules

Also, variable `XDG_DATA_DIRS` might need to be adjusted after installation to get full desktop integration.

If you install Gwyddion into `/usr/local` and get error message that `libgwyapp.so.0` cannot be found your system probably lacks standard library directories in the dynamic linker configuration. Notably, this happens on Ubuntu. Edit file `/etc/ld.so.conf` and add the line

```
/usr/local/lib
```

there.

Running

Running Gwyddion does not normally require any additional setup.

The misfeatures of some desktop environments, however, may render Gwyddion unusable and need to be disabled. The hijacking of program main menu in Unity makes most of Gwyddion menus inaccessible. It can be disabled by by unsetting `UBUNTU_MENUPROXY` while running Gwyddion:

```
UBUNTU_MENUPROXY= gwyddion
```

Deinstallation

Run

```
make uninstall
```

in the directory you previously compiled Gwyddion to remove it. If you have lost the source directory meanwhile you can try to unpack, configure and build it exactly as before and then issue **`make uninstall`**, although this relies on your ability to reproduce the build process.

RPM Packages

It is possible to build RPM packages on RPM-based GNU/Linux distributions directly from source code tarballs with

```
rpmbuild -tb gwyddion-2.49.tar.xz
```

where 2.49 is to be replaced with the actual version as above. This method was tested mainly on Fedora, openSuSE and Mandriva and the RPM spec file contains some specific provisions for these systems. Specific support for other RPM-based systems can be added on request.

6.3 Mac OS X

Much of the previous generic Unix/Linux installation section applies also to OS X. Therefore this section deals mainly with the specifics of OS X installation, some of the steps listed here are explained in more detail in the generic Unix section.

Beside building everything on your own (good luck), at this time there are two ways to install Gwyddion:

- using MacPorts (formerly Darwinports) and building from a Portfile.
- using Fink and installing Gwyddion using apt-get.
- using Homebrew and build Gwyddion using the brew formula.

Preparation

To install and run Gwyddion you need the Xcode Tools and X (SDK and App) installed. They where located on your CD-s/DVDs. The Xcode Tools where located on the first DVD as XcodeTools.mpkg below Xcode Tools, the X11SDK is located as X11SDK.pkg below the Packages Folder within Xcode Tools. X11 is localed as X11User.pkg below `System/Installation/Packa` even on the first DVD. If you have an CD Set the Discs may differ. The people from MacPorts recommending using the newest version of XCode. For further information look at the MacPorts Install Page. Also you should have some experience using Terminal.app. All the commands in the the rest of this section are to be entered and run in Terminal.app.

See installation dependencies section for an overview of required and optional packages to install prior to Gwyddion installation. The following table summarises how they are called in the two software collections:

Package	Fink	MacPorts
GTK+	gtk+2	gtk2
GtkGLExt	gtkglext1	gtkglext
FFTW3	fftw3	fftw-3
LibXML2	libxml2	libxml2

MacPorts

MacPorts is a Port based System for porting and installing Open Source/GNU software to OS X. It's based on using installation files called "Portfiles" which are describing the steps to compile and install an application. So it's far easy to port software to OS X using MacPorts but every computer has to compile the application. Get and install MacPorts. After you installed MacPorts, run

```
sudo port selfupdate
```

to update MacPorts to the latest version.

Usually installing ports with MacPorts is easy. But since X11 is not the native Desktop for OS X, things went a little worse. So it is recommended to install an alternative X11 before installing Gwyddion. The recommended alternatives are XQuartz on Leopard and the Port xorg-server on Tiger. After installing the suggested X11-System, Gwyddion can be then build and installed simply by

```
sudo port install gwyddion
```

To install xorg-server (Tiger) simply type

```
sudo port install xorg-server
```

this is *needed* for the 3D view on Tiger. After everything is done, you will find the StartUp-Icon below `/Applications/MacPorts`.

Fink

Get and install Fink. After you installed Fink run

```
apt-get update
```

to update the database of available packages and install Gwyddion with

```
apt-get install gwyddion
```

To install Gwyddion from source code, for instance if you want to install a development version, you need to install the required packages listed in the above table and then follow the generic Unix installation section instructions.

Running

On MacPorts you simply click on the StartUp-Icon and wait until Gwyddion appears. Using Fink or a self-compiled version you should follow the steps below: Start X11.app and type in Terminal.app

```
export DISPLAY=":0"
```

Then run Gwyddion from the folder it was installed to. This is typically `/usr/local/bin` for Fink. So for example for Fink run:

```
/usr/local/bin/gwyddion
```

You can also configure X11.app to run Gwyddion via: Locate X11.app in your dock, open the menu, choose Applications, choose Customize from the next menu. Here you can choose add and enter the name (Gwyddion for example) as *Menu Name* and the complete path to Gwyddion (e.g. /usr/local/bin/gwyddion) as *Command*. After this you can choose Gwyddion from the X11 menu.

6.4 Cross-Compiling for MS Windows

Cross-compiling Gwyddion for MS Windows under Linux is quite similar to normal Unix compilation with certain additional setup and extra steps. Although the process is quite smooth the initial setup may seem a bit complicated. If, in addition, you are not familiar with the normal Unix compilation you might wish to start with that and attempt cross-compilation once you familiarise yourself with the basic procedure.

These instructions describe compilation under Fedora using its MinGW cross-compilation support as this is what Gwyddion developers use. In general, the instructions work on the current version of Fedora. Compilation on other versions and other RedHat-based distributions (CentOS, Scientific Linux, ...) should be similar and relatively straightforward, possibly with some tweaks. Building on, for instance, openSUSE will require modifications. Reports of success (or failure) of cross-compilation of Gwyddion in other distributions and environments and namely improvements to these instructions are welcome.

Full cross-compilation has the following steps:

- configuration for mingw64/mingw32,
- compilation,
- installation into a staging area,
- creation of an installer using NSIS.

A script is available that automatically performs all the steps, as described below.

Setup

Before the first compilation you must set up the cross-compilation environment. This has to be done only once.

Base MinGW Packages

Run as root:

```
dnf install mingw{32,64}-{gcc-c++,gtk2,libxml2,minizip,fftw,gtkglext,libwebp,
    OpenEXR}
```

to install the necessary mingw32 and mingw64 packages. Several more packages will be installed as dependencies of those explicitly given here. Note that, technically, some of the packages are optional dependencies and you can build a MS Windows installer without them (after some adjustments). Neverhteless the standard installers include these packages and the cross-compilation scripts expect them to be present by default.

Gwyddion.net repository

MinGW versions of a few packages used by Gwyddion are not available in Fedora yet (or any more), or there may be some patches we would like to include but they are not in Fedora packages. This includes HDF5 and gtksourceview2 (only used by pygwy).

You can build these additional packages using the patches and spec files at https://sourceforge.net/projects/gwyddion/files/-mingw32-cross-compile/, however, it should be much easier to just install them using **dnf**. For this, download and install the gwyddion.net repository configuration package. The installation brings two repository configurations, for the program and for the extra MinGW packages. By default only the `gwyddion` repository is enabled. Enable `gwyddion-manually` by editing `/etc/yum.repos.d/gwyddion-mingw.repo` for the MinGW packages. Run the following command to install them:

```
dnf install mingw32-{hdf5,gtksourceview2}
```

Be warned the repo overrides a few Fedora MinGW packages to keep them at specific versions.

Wine

Wine is the MS Windows compatibility layer/emulator for Unix. It is used to run NSIS that creates the executable Gwyddion Windows installer. Wine can also be used to run and test the cross-compiled Gwyddion, as described below.

Run

```
dnf install wine
```

to install Wine.

NSIS

Nullsoft scriptable install system (NSIS) is used to create the Gwyddion installer. This is a MS Windows program, therefore, it is installed *under Wine*. A cross-compiled version of NSIS might be available in the distribution but we have found the original more reliable.

Download NSIS from its web page and run

```
wine nsis-3.06-setup.exe
```

replacing 3.06 with the actual version. Version 3.06 is the oldest NSIS 3 that has been tested, but any version probably works.

Python

To compile pygwy you need to install Python into Wine. The steps are the same as if you just want to use pygwy, except that all packages listed in Enabling pygwy need to be installed using **msiexec**:

```
wine msiexec /i python-2.7.16.msi
wine msiexec /i pygobject-2.28.3.win32-py2.7.msi
wine msiexec /i pycairo-1.8.10.win32-py2.7.msi
wine msiexec /i pygtk-2.24.0.win32-py2.7.msi
```

or similarly.

Support scripts

Support scripts and data are available in `mingw32-cross-compile` module in the Gwyddion subversion repository. Run

```
svn checkout http://svn.code.sf.net/p/gwyddion/code/trunk/mingw32-cross-compile
```

to check it out from the repository.

The most important tool you obtain is the **cross-build-32** (or **cross-build-64**) script that automates all the cross-compilation steps. Before you use it for the first time, review file `setup32` that defines where various things are located (or `setup64` for the 64-bit target). The default contents looks as follows:

```
source_dir=$HOME/Projects/Gwyddion/gwyddion-mingw
mingw_prefix=/usr/i686-pc-mingw32/sys-root/mingw
target_prefix=$HOME/opt/gwyddion-mingw32
python_dir=$HOME/.wine/drive_c/Python27
nsis_compiler=C:\\Program\ Files\ \(x86\)\\NSIS\\makensis.exe
```

Variable `source_dir` specifies the location of the unpacked or checked-out Gwyddion source code and it will likely need to be adjusted. Variable `target_prefix` specifies the installation directory (staging area) for the cross-compiled Gwyddion. The default value should be reasonable and you do not need to change it unless you want to. The remaining variables, `mingw32_prefix`, `nsis_compiler` and `python_dir`, specify the location of MinGW files, NSIS compiler and Win32 Python, respectively. They do not need to be changed from the default values under normal circumstances although NSIS can be installed in either `Program Files (x86)` or `Program Files` by default depending on Wine configuration. Note `setup` is read by shell so there must not be any spaces around =.

Compilation

The setup was tedious but it was worth it because the compilation is then extremely simple. Run

```
./cross-build-32
```

in `mingw32-cross-compile` directory to build Win32 insstaller. That's all. If it succeeds an executable Gwyddion Windows installer with bundled GTK+ and everything will be created in `$target_prefix`. Similarly, the Win64 installer is built just with

```
./cross-build-64
```

You can make a coffee meanwhile – or study the `cross-build` script (it is actually quite short and clear).

Note the cross-build scripts run `autogen.sh` but do not clean the source code directory. You may wish to do that manually if you compile Gwyddion repeatedly. Especially if you build for both architectures in the same directory, make sure to run

```
make distclean
```

between the builds to get the source directory back to a well-defined state.

Running under Wine

Compiled Gwyddion can be run under Wine. Assuming the default value of `target_prefix`:

```
wine ~/opt/gwyddion-mingw32/bin/gwyddion.exe
```

To run `gwyddion.exe` the dynamic linker must be able to find all the necessary DLLs. This is ensured in a somewhat crude way by script **copysysfiles** that copies all necessary MinGW files from system to `$target_prefix`. Since **copysysfiles** is executed by **cross-build** you normally do not need to execute it manually.

The second step that might be necessary is setting registry key

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\App Paths\gwyddion. ↩
    exe
```

to point to `gwyddion.exe` and the value `Path` to point to the `bin` subdirectory.

Cross-compilation of standalone modules

Cross-compilation of standalone modules requires only Gwyddion headers and Win32 development libraries. While they can be surely obtained by cross-compiling entire Gwyddion is it not necessary to do so. By compiling only the libraries you can avoid the installation of various rarer Gwyddion dependencies. This can be achieved using the patch `gwyddion-2.22-build-only-libs.pa` available among the build scripts.

But it gets even easier, the MinGW Gwyddion libraries are available as a RPM package `mingw32-gwyddion-libs` in the gwyddion.net repository.

Once you have this package installed you can try cross-compilation of the sample standalone module `threshold-example` that is available in the subversion repository (or as a tarball). See `README` therein for some more details.

6.5 Compiling on MS Windows using MinGW

Although the standard MS Windows executables are created using cross-compilation it is also possible to build Gwyddion on MS Windows using the MinGW-W64 port of GNU tools to MS Windows. The standard MS Windows excutables also come with almost all the optional features packaged – the notable exception being Python scripting. Getting all these components work in MS Windows requires additional effort. However, the most likely reason for compiling on MS Windows is to obtain all the necessary files to develop and build standalone Gwyddion modules and for this purpose building all the optional components is not necessary.

See the MinGW-W64 website for its installation instructions. The procedure is then essentially the same as the normal Unix compilation. Some MinGW-specific remarks follow.

Dependencies can be install using **pacman**, for instance

```
pacman -S mingw-w64-ucrt-x86_64-gtk2
pacman -S mingw-w64-ucrt-x86_64-fftw
```

for GTK+2 and FFTW.

It has also been reported that the GTK+ 2.24.10 bundle can be successfully used. After installing it, set in the MSYS shell

```
PKG_CONFIG=PATH-TO-GTK+/gtk+/bin/pkg-config.exe
```

where `PATH-TO-GTK+` needs to be replaced with the actual GTK+ installation directory.

To compile only the libraries, it may be useful to use the patch `gwyddion-2.22-build-only-libs.patch` described in the cross-compilation section. In addition, it seems that the MinGW libintl redefines `printf()` to `libintl_printf()` which it, however, does not provide. This leads to link failure of `gwyddion.exe`. This can be 'fixed' by simply removing `include/libintl.h` in the GTK+ directory.

6.6 Compiling on MS Windows using Microsoft Visual Studio

Bundle Installation

Bundle for compiling in Visual Studio is available here.

Solution

Copy `msvc2015` folder to **gwyddion** source code folder (`/trunk/gwyddion` folder in `svn`, simply called `gwyddion` in the following text).

Generated files

Copy contents of `generated-files` folder to `gwyddion` folder.

Python 2.7

Install Python 2.7. Version 2.7.11 does not work. The latest stable version is Python 2.7.9. Python installation packs are in `libs-install` folder. Install them to `C:\libs\Python\Win32` and `C:\libs\Python\x64` folders. It is unpossible to link against Python Debug library because `python27_d.lib` file is missing. It does not help to define `MS_NO_COREDLL`, `Py_NO_ENABLE_SHARED` and `SWIG_PYTHON_INTERPRETER_NO_DEBUG`.

Resolution:

- copy `python27.lib` to `python27_d.lib`
- in `pyconfig.h` comment out `define Py_DEBUG`

`python27_d.lib` and `pyconfig.h` are in `libs\Python27` folder so you can just copy them to `C:\libs\Python27` folder.

Libraries

Gwyddion depends on external libraries. Set paths to libraries as **environment variables**. Path must be ended by '\'. Stable versions of these libraries are in `libs` folder. Example of environment variables setting follows:

```
CFITSIO_DIR = C:\libs\cfitsio\
GTK_DIR = C:\libs\Gtk\
GTKGLEXT_DIR = C:\libs\GetGLExt\
LIBICONV_DIR = C:\libs\libiconv\
LIBXML_DIR = C:\libs\libxml\
PYTHON_DIR = C:\libs\Python27\
ZLIB_DIR = C:\libs\zlib\
```

'Build Events'

A part of project build is copying all files needed to run Gwyddion. Specific files are copied from `gwyddion` folder to `$(OutDir)` folder (or appropriate subfolder). DLL files are copied from extern library paths defined as environment variables to `$(OutDir)` folder. Commands are in menu item `Project/Properties/Build Events/Post-Build Event`.

Solution Generating

Gwyddion is natively compiled on Linux using **autotools**. Conversion of conditional compilation prescriptions between different platforms is not trivial (conditions, dependencies, variable expansion, different configuration files on Linux/Windows due different library availability). These factors avoid adequate full automatic conversion.

`gen-gwyddion-msvc-sln.py` is script generating Visual Studio **solution** and **projects**. Script must be run in `gwyddion` folder (see above). **Gwyddion** must be compiled on Linux before running the script. Linux compilation generates specific .c, .h files and allows script to generate exported function files (.def). All these files are needed for compilation on Windows. They are all sorted to `generated-files` folder and must be copied to `gwyddion` folder on Windows. The script results in Visual Studio `solution` and `projects` which are generated to `msvc2015` folder. All folders generated by script are created in `gwyddion` folder.

`config.h` a `gwyconfig.h` configuration files are natively generated on Linux by **autotools**. These files correspond to Linux configuration and must be modified to be able to compile in Visual Studio. Prepared Windows friendly configuration files are part of script. These files are copied to `generated-files` folder during script generating process.

Procedure:

Compile on Linux

On Linux. Run `./autogen.sh` and `make` in source code folder (usually `gwyddion` folder). See compilation on Linux/Unix.

Run script to generate `solution` and to copy out generated files

On Linux. Run `gen-gwyddion-msvc-sln.py` in `gwyddion` folder.

Generated folder	Contents
`msvc2015`	Gwyddion Solution
`msvc2015/generated-files`	.def, .c, .h files

Copy all files and folders from `./msvc2015/generated-files` on Linux machine to `./gwyddion` on Windows machine.

Installation

On Windows. See the bundle installation section.

6.7 Subversion Checkout, Development

Gwyddion uses Subversion version control system for source code revision management. The organisation of the repository is described on project web pages. For example the current head revision of the program itself can be checked out with

```
svn checkout http://svn.code.sf.net/p/gwyddion/code/trunk/gwyddion
```

The repository does not contain any generated files, no matter how exotic tools may be necessary to generate them. Therefore, additional packages are required for building from a fresh checkout. There are also certain platform limitations. The additional tools and packages required for development are essentially the same as for compilation from Subversion checkout. More precisely, to build from a fresh checkout all the additional tools are necessary, whereas development may require only a subset of them or even none, depending on the type and extent of the changes in the source code.

Additional development build dependencies (many optional if you start from source tarballs instead of a fresh checkout and do not do heavy modifications):

- GNU autoconf $\geq$ 2.60
- GNU automake $\geq$ 1.11
- GNU libtool $\geq$ 1.4
- Python 2 or 3
- gtk-doc $\geq$ 1.12
- GNU gettext $\geq$ 0.12, including development stuff
- Inkscape $\geq$ 0.91
- xsltproc
- Epydoc
- pngcrush $\geq$ 1.8.9
- probably GNU versions of most tools: the compiler, binutils, ...

After a fresh checkout, run `./autogen.sh` with any arguments you would give to **configure**. Note it automatically adds options `--enable-maintainer-mode` and `--enable-gtk-doc` to ensure the rules for creation and updates of various files are active. Generally, you should always use `--enable-maintainer-mode` if you intend to change the program source code in a non-trivial way.

On some systems, **autogen.sh** can fail even if you have sufficient versions of autotools installed. These systems do not install general **autoconf** or **automake** commands, only versioned commands such as **autoconf261** or **automake19**. This makes it particularly difficult to find for example "**automake** 1.9 or newer" with no limit on how newer it can be. Therefore, **autogen.sh** does not attempt this at all. You can either create unversioned symbolic links to the versioned commands or run **autogen.sh** as follows: `AUTOCONF=autoconf261 AUTOHEADER=autoheader261 ./autogen.sh` You may need to set the following variables: `ACLOCAL`, `AUTOCONF`, `AUTOHEADER`, `AUTOM4TE`, `AUTOMAKE`, `LIBTOOLIZE`. In addition, some operating systems may install **autoconf** macros in a place **aclocal** does not find them by default. This can be fixed by setting variable `ACLOCAL_FLAGS` to give **aclocal** additional search paths: `ACLOCAL_FLAGS="-I /usr/local/share/aclocal" ./autogen.sh`

It is often necessary to combine these adjustments. For instance on FreeBSD, where all tools are versioned, one typically invokes (broken to lines for easier reading):

```
AUTOCONF=autoconf261 \
AUTOHEADER=autoheader261 \
AUTOM4TE=autom4te261 \
AUTOMAKE=automake19 \
ACLOCAL=aclocal19 \
ACLOCAL_FLAGS="-I /usr/local/share/aclocal" \
CPPFLAGS=-I/usr/local/include \
LDFLAGS=-L/usr/local/lib \
./autogen.sh --prefix=...
```

If **autogen.sh** passes you can compile the program as usual.

MS Windows

Since the standard method to create MS Windows executables is cross-compilation in Linux the recommended method to develop for MS Windows is also to compile in Linux. This can be done either on a different physical computer using **ssh** or in a virtual machine running on the same computer as the host MS Windows system. In both cases the Gwyddion build directory (and other directories) can be shared between the Linux and MS Windows systems using either Samba or a shared directory mechanism of the virtual machine and the compiled executables thus can be directly tested in MS Windows without having to transfer files back and forth.

6.8 Developing Gwyddion

You are encouraged to become a developer of Gwyddion.

If you want to become developer, we recommend you to start with some simple modules (see module tutorial), to see how the application works. If you write a module or plug-in, you are encouraged to share it here with other Gwyddion users. Let us know, so that we can link your modules or plug-ins to these pages or even include in it Gwyddion. You don't have to limit yourself to modules or plug-ins of course, but they should be easier to start with.

API References

There are many functions that can help you while developing your module. See API reference at Documentation section of the project web.

Bug reports

We will be very happy if you send us bug reports if you find errors in Gwyddion. For doing this, please, specify as much as possible the situation that led to error, operating system and Gwyddion version used. You can also send us the SPM data that were being processed when the problem was found, this is necessary namely for reporting bugs related to file loading and saving.

The preferred bug reporting method is to send an e-mail to klapetek@gwyddion.net.

Appendix A

GNU General Public License

A.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

A.2 Terms And Conditions For Copying, Distribution And Modification

Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License.

Exception:
If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

A.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>    <name of author>
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author Gnomovision
comes with ABSOLUTELY NO WARRANTY; for details type "show w".
This is free software, and you are welcome to redistribute it under
certain conditions; type "show c" for details.
```

The hypothetical commands “show w” and “show c” should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than “show w” and “show c”; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
program “Gnomovision” (which makes passes at compilers) written
by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B

GNU Free Documentation License

B.1 Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

B.2 Applicability And Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing

tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

B.3 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

B.4 Copying In Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

B.5 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

GNU FDL MODIFICATION CONDITIONS

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

B.6 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

B.7 Collections Of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

B.8 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

B.9 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

B.10 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

B.11 Future Revisions Of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

B.12 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

SAMPLE INVARIANT SECTIONS LIST

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

SAMPLE INVARIANT SECTIONS LIST

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Chapter 7

Index