Last modification: 31.03.15

Graphical User Interface for GAMMA

*John Truckenbrodt <john.truckenbrodt@uni-jena.de>*

# Introduction

The document at hand is intended as a practical guideline to the graphical user interface (GUI) developed within the practical student project of the GAMMA software course held at the Friedrich-Schiller-University Jena in 2014/15.

The aim of this project was to develop a modularized SAR image processing workflow using the software GAMMA, comprising major tasks like raw data import, multilooking, geocoding, image coregistration, interferogram generation, coherence estimation, KML generation and display functionalities.

The developed modular script package was written in Python as this language was the most commonly used amongst the course participants and its easily understandable syntax alleviates the chance of further development by future courses.

The GUI described in this document can be seen as an attempt to centralize repeatedly executed tasks, like text file reading/writing and file search, to give less scripting-experienced SAR image users the opportunity to use the software GAMMA and to accelerate the execution of routinely used GAMMA commands.

In the following sections the general layout of the GUI and its processing capabilities are briefly described from a menu navigation perspective. Please refer to the individual scripts for technical details.

# Main Menu

On execution of the main script *gammaGUI.py,* the starting menu of the GUI is opened. Here the user is asked for a working directory in which all produced files are saved. Without having defined such directory, selecting the buttons in the top menu bar will throw an error as the underlying processes will only operate in this directory.

The menu bar comprises four submenus, whose names originate from the GAMMA processing task categorization: MSP (Modular SAR Processor), ISP (Interferometric SAR Processor), GEO (Geocoding) and DISP (Display Tools and Utilities).

# Meta Data

## Parameter files

While using the GUI, numerous GAMMA commands are executed, which require many parameters. These parameters are entered via interactive dialog windows. The central script for this functionality is *dialog.py*. On opening such dialog (by selecting a processing command from the menu) default parameters are supplied. Initially they are selected from the class *Main* in script *gammaGUI.py*. On execution of the command, these parameters are first written to a parameter file in a (newly created) subfolder PAR of the working directory. The name will be concatenated from the name of the executed script without extension and a suffix “.par”, i.e. *coreg.par* for script *coreg.py*.

In case the parameter file for the selected command already exists, the parameters are not read from class *Main*, but from the file instead. This way, preferential parameters can be stored directly in a newly created working directory and implemented defaults be overridden.

## Log files

GAMMA prints command results to the screen. This information is crucial for assessing the plausibility of the selected processing parameters. Since long screen prints are not desired within the GUI, this information is piped to log files. Such file, whose name will contain the GAMMA command name and the suffix “.log” (e.g. *geocode.log*) will be saved into a subfolder LOG of the working directory, which itself will contain further subfolders referring to the earlier mentioned processing categorization (i.e. ISP, GEO, etc.).

In case a command is executed repeatedly, the new information is appended to the existing file. This is managed by the function *run* of script *ancillary.py*.

## Errors, Warnings, Notifications

Please note, that the presented software is only complementary to the command line based software GAMMA. Although major command line printouts are redirected to text files, general information is still displayed. Hence, the command line is to be observed for information on the images being processed and error messages during the execution of the GUI. Notification and Error message windows will be displayed, yet they will not show any detailed information.

# IMP: Data Import Utility

This is the raw data import utility, which is based on the script *reader.py* (Stefan Engelhardt). On selection, an opening window gives the opportunity to define an import directory and a SAR sensor. The sensor dropdown menu gives the opportunity to only import data from the selected sensor. If this field is left blank data from all (currently implemented) sensors is imported.

On execution, the import directory and its subdirectories are scanned for files, which are passed to the script *reader.py*. If the file meets the specific requirements, the script converts the dataset to a GAMMA internal format. These newly created files are stored in a temporary directory. The parameter file is read for the sensor and the time stamp from which a new scene folder is created. The data is then moved from the temporary directory into the newly created folder.

The working directory is intended to only contain image data from the same geographical area. A warning will be given if the center latitudes or longitudes of the imported scenes differ by more than one degree.

# ISP: Interferometric SAR Processor

The following steps are to be performed in sequential order. Coregistration is required for all interferometric processing steps, multilooking prior to coherence estimation.

## SLC Coregistration

On selection the dialog requires two SLC files, primary and secondary. The secondary SLC will be registered to the primary SLC by the script *coreg.py* (Soner Üreyen, John Truckenbrodt). The created offset files are stored in a subdirectory ISP of the working directory for further refinement of the coregistration (not implemented) and use by subsequent interferometric processing. The registered and resampled SLC (RSLC) will be stored in the same folder as the secondary SLC (its original version).

## Multilooking

On selection, the window dialog asks for a target resolution of the resulting multilooked images (MLIs). The default option is “*automatic*”. In this case the script *multilook.py* (Elias Wolf, John Truckenbrodt) will calculate the smallest factors for creating (close to) square pixels and perform multilooking accordingly. In case a numeric target resolution is entered, the factors will be multiplied to approximate this resolution.

This is performed for all SLCs and RSLCs in the working directory, the outputs are named with the suffixes “\_mli” and “\_rmli” respectively and are stored in the same directory as their corresponding complex image.

## Interferogram Generation

A primary SLC and a secondary RSLC are required. Furthermore the offset file created by coregistration must exist in the subfolder ISP. If this is not the case, the corresponding script *interferogram.py* (Christina Limmer, John Truckenbrodt) will throw an error and the command is aborted.

The resulting interferogram (suffix “\_int”) will be stored in the subfolder ISP of the working directrory.

## Baseline Estimation

A primary and a secondary RSLC need to be supplied. For executing the script *baseline.py* (Christina Limmer, John Truckenbrodt), an offset parameter file and an interferogram must already exist for the selected SLC-RSLC combination. If this is not the case, an error is thrown. The resulting file, carrying the suffix “\_base” will be stored in the folder ISP.

## Interferogram Flattening

The corresponding script *ph\_slope\_base.py* (Axel Schmidt, John Truckenbrodt) requires no further parameters. On execution all existing interferograms are searched for and flattened. Errors are thrown if either the offset file or the baseline file are missing.

The results are stored in the same folder as the interferogram and are identified by the suffix “\_flt”.

## Coherence Estimation

The user can choose from two implemented approaches:

* cc\_wave: the classical approach
* cc\_ad: adaptive coherence estimation with consideration of phase slope and texture

The corresponding scripts *coherence\_wave.py* and *coherence\_ad.py* (Axel Schmidt, John Truckenbrodt) will first scan the working directory for flattened interferograms (with a suffix “\_flt”). In case such file is found, the names for the MLI (primary) and RMLI (coregistered MLI, secondary) are concatenated from the corresponding time stamps (which are contained in the name of the FLT) and searched for in the working directory. If these two images do not exist an error will be thrown. The number of range samples is retrieved by reading the parameter text file of the MLI.

The resulting files are saved in the subdirectory ISP and are identified by their suffixes “\_cc\_ad” and “\_cc\_wave”.

# GEO: Geocoding

## SRTM Download

The corner coordinates of all SLC (single look complex) scenes will be extracted and all SRTM tiles, which overlap with these coordinates, are downloaded from the USGS server ([*http://dds.cr.usgs.gov/srtm/version2\_1/SRTM3/*](http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/)) to a subfolder *DEM* of the working directory. The received zip archives are then extracted and deleted. This is done by the function *srtm* of the script *ancillary.py*.

## DEM Preparation

This is the interface for the script *dem\_procesing.py* (Soner Üreyen, John Truckenbrodt). The command automatically performs the following tasks:

* GAMMA DEM parameter file generation (defining general DEM meta data)
* Mosaicking if more than one SRTM tile has been downloaded
* Replacement and interpolation of missing data
* Transformation to UTM and resampling to desired resolution using SLC parameter file for geographical information

The subdirectory DEM of the working directory is scanned for SRTM files with the suffix “.hgt”, the use of other DEMs is currently not implemented. For retrieving the SLC projection, the working directory is scanned for SLC files and the first entry is used as reference. Names of the created files:

* Initial mosaic: *mosaic*
* Mosaic with replaced and interpolated values: *dem\_final*
* *dem\_final* transformed to UTM: *dem\_final\_utm*

Errors are thrown if no DEM and SLC files are found.

## Geocoding

On execution this script will geocode all MLIs (suffix “\_mli”) and all coherence images (suffixes “\_cc\_ad” and “cc\_wave”) by running the script *geocoding.py* (Soner Üreyen, John Truckenbrodt).

First all these files are searched for and grouped by their directory. It is assumed, that all images in one directory belong to the same acquisition, i.e. they are coregistered to each other.

From a priority list the master is selected and all necessary GAMMA commands are executed. The intermediate results are stored into a subfolder GEO of the directory containing the master image.

In case the command *init\_offsetm* reports a SNR (signal to noise ratio) below the defined threshold, a message is reported to the console and the loop will continue with the next scene group.

The last command of the process is *geocode\_back*. This will perform the actual geocoding from the created lookup table (LUT). This is performed for all images in the scene group. The resulting files with the suffix “\_geo” will be stored in the same folders as the original image.

## BMP and KML generation

This will automatically create .bmp and .kml files for all existing images with a suffix “\_geo”.

In order for the .bmp and .kml to function together the \_geo image must be in equiangular projection (EQA). Geocoding to EQA is currently not implemented. Thus, being loaded into Google Earth, the KML will be displayed without the actual image.

# DISP: Display Utilities

A selection of GAMMA image displays is implemented:

* Intensity: *dispwr, dis2pwr*
* SLC: *disSLC, dis2SLC*
* Coherence: *discc*
* Height: *disdem\_par*

For all commands except SLC display, the display width is read from the parameter file, i.e. the full image width is displayed. For displaying SLC files, the number of samples can be entered via the GUI.