WhiteboxTools User Manual

John B. Lindsay, PhD, University of Guelph

Contents

WhiteboxTools Version 0.6

Geomorphometry and Hydrogeomatics Research Group
University of Guelph
Guelph, Canada
April 3, 2018

GHRG
Geomorphometry & Hydrogeomatics Research Group

1. Introduction

WhiteboxTools is an advanced geospatial data analysis platform developed by Prof. John Lindsay (webpage; jblindsay) at the University of Guelph's Geomorphometry and Hydrogeomatics Research Group (GHRG). The project began in January 2017 and quickly evolved in terms of its analytical capabilities. *WhiteboxTools* can be used to perform common geographical information systems (GIS) analysis operations, such as cost-distance analysis, distance buffering, and raster reclassification. Remote sensing and image processing tasks include image enhancement (e.g. panchromatic sharpening, contrast adjustments), image mosaicking, numerous filtering operations, simple classification (k-means clustering), and common image transformations. *WhiteboxTools* also contains advanced tooling for spatial hydrological analysis (e.g. flow-accumulation, watershed delineation, stream network analysis, sink removal), terrain analysis (e.g. common terrain indices such as slope, curvatures, wetness index, hillshading; hypsometric analysis; multi-scale topographic position analysis), and LiDAR data processing. LiDAR point clouds can be interrogated (LidarInfo, LidarHistogram), segmented, tiled and joined, analyzed for outliers, interpolated to rasters (DEMs, intensity images), and ground-points can be classified or filtered. *WhiteboxTools* is not a cartographic or spatial data visualization package; instead it is meant to serve as an analytical backend for other data visualization software, mainly GIS.

> In this manual, ***WhiteboxTools*** refers to the standalone geospatial analysis library, a collection of tools contained within a compiled binary executable command-line program and the associated Python scripts that are distributed alongside the binary file (e.g. *whitebox_tools.py* and *wb_runner.py*). ***Whitebox Geospatial Analysis Tools*** and ***Whitebox GAT*** refer to the GIS software, which includes a user-interface (front-end), point-and-click tool interfaces, and cartographic data visualization capabilities.

Although *WhiteboxTools* was first developed with to serve as a source of plugin tools for the *Whitebox Geospatial Analysis Tools (GAT)* open-source GIS project, the tools contained in the library are stand-alone and can run outside of the larger *Whitebox GAT* project. See Interacting With *WhiteboxTools* From the Command Prompt for further details. There have been a large number of requests to call *Whitebox GAT* tools and functionality from outside of the *Whitebox GAT* user-interface (e.g. from Python automation scripts). *WhiteboxTools* is intended to meet these usage requirements. For example, a *WhiteboxTools* plug-in for QGIS is available.

The current version of *Whitebox GAT* contains many equivelent tools to those found in the *WhiteboxTools* library, although they are developed using the Java programming language. A future version of *Whitebox GAT* will replace these previous tools with the new *WhiteboxTools* backend. This transition will occur over the next several releases. Eventually most of the approximately 450 tools contained within *Whitebox GAT* will be ported to *WhiteboxTools*. In addition to separating the processing capabilities and the user-interface (and thereby reducing the reliance on Java), this migration should significantly improve processing efficiency. This is because Rust, the programming language used to develop *WhiteboxTools*, is generally faster than the equivalent Java code and because many of the *WhiteboxTools* functions are designed to process data in parallel wherever possible. In contrast, the older Java codebase included largely single-threaded applications.

In addition to *Whitebox GAT*, the *WhiteboxTools* project is related to other GHRG software projects including, the *GoSpatial* project, which has similar goals but is designed using the Go programming language instead of Rust. *WhiteboxTools* has however superseded the *GoSpatial* project, having subsumed all of its functionality. *GoSpatial* users should now transition to *WhiteboxTools*.

2. Downloads and Installation

WhiteboxTools is a stand-alone executable command-line program with no actual installation. Simply download the appropriate file for your system and decompress the folder. Pre-compiled binaries can be downloaded from the *Geomorphometry and Hydrogeomatics Research Group* software web site for various supported operating systems. Depending on your operating system, you may need to grant the *WhiteboxTools* executable file execution privileges before running it. If you intend to use the Python programming interface for *WhiteboxTools* you will need to have Python 3 installed.

It is likely that *WhiteboxTools* will work on a wider variety of operating systems and architectures than those of the distributed pre-compiled binaries. If you do not find your operating system/architecture in the list of available *WhiteboxTool* binaries, then compilation from source code will be necessary. WhiteboxTools can be compiled from the source code with the following steps:

1. Install the Rust compiler; Rustup is recommended for this purpose. Further instruction can be found at this link.

2. Download the *Whitebox GAT* source code. Note: *WhiteboxTools* is currently housed as a sub-repository of the main *Whitebox GAT* repo. To download the code, click the green Clone or download button on the GitHub repository site.

3. Decompress the zipped download file.

4. Open a terminal (command prompt) window and change the working directory to the whitebox_tools sub-folder, which is contained within the decompressed downloaded Whitebox GAT folder:

```
>> cd /path/to/folder/whitebox_tools/
```

5. Finally, use the rust package manager Cargo, which will be installed alongside Rust, to compile the executable:

```
>> cargo build --release
```

Depending on your system, the compilation may take several minutes. When completed, the compiled binary executable file will be contained within the *whitebox_tools/target/release/ folder*. Type *./whitebox_tools --help* at the command prompt (after changing the directory to the containing folder) for information on how to run the executable from the terminal.

> The '>>' is shorthand used in this document to denote the command prompt and is not intended to be typed.

Be sure to follow the instructions for installing Rust carefully. In particular, if you are installing on Microsoft Windows, you must have a linker installed prior to installing the Rust compiler (*rustc*). The Rust webpage recommends either the **MS Visual C++ 2015 Build Tools** or the GNU equivalent and offers details for each installation approach. You should also consider using **RustUp** to install the Rust compiler.

3. Supported Data Formats

The *WhiteboxTools* library can currently support reading/writing raster data in GeoTIFF (.tif), *Whitebox GAT*(.tas and .dep), ESRI (ArcGIS) ASCII (.txt) and binary (.flt and .hdr), GRASS GIS, Idrisi (.rdc and .rst), SAGA GIS (binary–.sdat and .sgrd–and ASCII formats), and Surfer 7 (.grd) data formats. *The BigTIFF (64-bit) format is not currently supported.* The library is primarily tested using Whitebox raster and GeoTIFF data sets and if you encounter issues when reading/writing data in other formats, you should report the issue. Please note that there are no plans to incorporate third-party libraries, like GDAL, in the project given the design goal of keeping a pure (or as close as possible) Rust codebase without third-party dependencies. This design greatly simplifies installation of the library.

Please note that throughout this manual code examples that manipulate raster files all use the GeoTIFF format (.tif) but any of the supported file extensions can be used in its place.

At present, there is limited ability in *WhiteboxTools* to work with vector geospatial data. Shapefiles geometries (.shp) and attributes (.dbf) can be read and some tools take vector inputs. There is currently no support for writing vector data although this feature is being actively developed. Other vector data formats may be added in the future.

LiDAR data can be read/written in the common LAS data format. *WhiteboxTools* can read and write LAS files that have been compressed (zipped with a .zip extension) using the common DEFLATE algorithm. Note that only LAS file should be contained within a zipped archive file. The compressed LiDAR format LAZ and ESRI LiDAR format are not currently supported by the library. The following is an example of running a LiDAR tool using zipped input/output files:

```
>>./whitebox_tools -r=LidarTophatTransform -v --wd="/path/to/data/"
-i="input.las.zip" -o="output.las.zip" --radius=10.0
```

Note that the double extensions (.las.zip) in the above command are not necessary and are only used for convenience of keeping track of LiDAR data sets (i.e. .zip extensions work too). The extra work of decoding/encoding compressed files does add additional processing time, although the Rust compression library that is used is highly efficient and usually only adds a few seconds to tool run times. Zipping LAS files frequently results 40-60% smaller binary files, making the additional processing time worthwhile for larger LAS file data sets with massive storage requirements.

4. Interacting With *WhiteboxTools* From the Command Prompt

WhiteboxTools is a command-line program and can be run either by calling it from a terminal application with appropriate commands and arguments, or, more conveniently, by calling it from a script. The following commands are recognized by the *WhiteboxTools* library:

Command	**Description**
--cd, --wd	Changes the working directory; used in conjunction with --run flag.
-h, --help	Prints help information.
-l, --license	Prints the whitebox-tools license.
--listtools	Lists all available tools, with tool descriptions. Keywords may also be used, --listtools slope.
-r, --run	Runs a tool; used in conjunction with --cd flag; -r="LidarInfo".
--toolbox	Prints the toolbox associated with a tool; --toolbox=Slope.
--toolhelp	Prints the help associated with a tool; --toolhelp="LidarInfo".
--toolparameters	Prints the parameters (in json form) for a specific tool; e.g. --toolparameters="FeaturePreservingDenoise".
-v	Verbose mode. Without this flag, tool outputs will not be printed.
--viewcode	Opens the source code of a tool in a web browser; --viewcode="LidarInfo".
--version	Prints the version information.

Generally, the Unix convention is that single-letter arguments (options) use a single hyphen (e.g. -h) while word-arguments (longer, more descriptive argument names) use double hyphens (e.g. --help). The same rule is used for passing arguments to tools as well. Use the *--toolhelp* argument to print information about a specific tool (e.g. --toolhelp=Clump).

> Tool names can be specified either using the snake_case or CamelCase convention (e.g. *lidar_info* or *LidarInfo*).

The following is an example of calling the *WhiteboxTools* binary executable file directly from the command prompt:

```
>>./whitebox_tools --wd='/Users/johnlindsay/Documents/data/' ^
--run=DevFromMeanElev --input='DEM clipped.tif' ^
--output='DEV raster.tif' -v
```

Notice the quotation marks (single or double) used around directories and filenames, and string tool arguments in general. After the `-\-run` flag, used to call a tool, a series of tool-specific flags are provided to indicate the values of various input parameters. Note that the order of these flags is unimportant. Use the '-v' flag (run in verbose mode) to force the tool to print output to the command prompt. Please note that the whitebox_tools executable file must have permission to be executed; on some systems, this may require setting special permissions. Also, the above example uses the forward slash character (/), the directory path separator used on unix based systems. On Windows, users should use the back slash character (\) instead. Also, it is sometimes necessary to break (ˆ) commands across multiple lines, as above, in order to better fit with the documents format. Actual command prompts should be contained to a single line.

5. Interacting With *WhiteboxTools* Through Python Scripting

By combining the *WhiteboxTools* library with a high-level scripting language, such as Python, users are capable of creating powerful stand-alone geospatial applications and workflow automation scripts. In fact, *WhiteboxTools* functionality can be called from many different programming languages. However, given the prevalent use of the Python language in the geospatial fields, the library is distributed with several resources specifically aimed at Python scripting. This section focuses on how Python programming can be used to interact with the *WhiteboxTools* library.

> Note that all of the following material assumes the user system is configured with Python 3. The code snippets below are not guaranteed to work with older versions of the language.

5.1 Using the *whitebox_tools.py* script

Interacting with *WhiteboxTools* from Python scripts is easy. To begin, each script must start by importing the *WhiteboxTools* class, contained with the *whitebox_tools.py* script; a new `WhiteboxTools` object can then be created:

```
1 from whitebox_tools import WhiteboxTools
2
3 wbt = WhiteboxTools()
```

Depending on the relative location of the *WhiteboxTools* directory and the script file that you are importing to, the import statement may need to be altered slightly. In the above script, it is assumed that the folder containing the *WhiteboxTools* files (including the *whitebox_tools* Python script) is named `WBT` (Line 1) and that the calling script is located in the parent directory of WBT. See An Example WhiteboxTools Python Project for more details on project set-up. The use of `wbt` to designate the WhiteboxTools object variable in the above script (Line 3) is just the convention used in this manual and other project resources. In fact, any variable name can be used for this purpose.

The `WhiteboxTools` class expects to find the *WhiteboxTools* executable file (*whitebox_tools.exe* on Windows and *whitebox_tools* on other platforms) within the same directory (`WBT`) as the *whitebox_tools.py* script. If the binary file is located in a separate directory, you will need to set the executable directory as follows:

```
wbt.set_whitebox_dir('/local/path/to/whitebox/binary/')
# Or alternatively...
wbt.exe_path = '/local/path/to/whitebox/binary/'
```

Individual tools can be called using the convenience methods provided in the `WhiteboxTools` class:

```
# This line performs a 5 x 5 mean filter on 'inFile.tif':
wbt.mean_filter('/file/path/inFile.tif', '/file/path/outFile.tif', 5, 5)
```

Each tool has a cooresponding convenience method. The listing of tools in this manual includes information about each tool's Python convienience method, including default parameter values. Parameters with default values may be optionally left off of function calls. In addition to the convenience methods, tools can be called using the `run_tool()` method, specifying the tool name and a list of tool arguments. Each of the tool-specific convenience methods collect their parameters into a properly formated list and then ultimately call the `run_tools()` method. Notice that while internally *whitebox_tools.exe* uses CamelCase (e.g. MeanFilter) to denote tool names, the Python interface of *whitebox_tools.py* uses snake_case (e.g. mean_filter), according to Python style conventions. The only exceptions are tools with names that clash with Python keywords (e.g. `And()`, `Not()`, and `Or()`).

The return value can be used to check for errors during operation:

```
if wbt.ruggedness_index('/path/DEM.tif', '/path/ruggedness.tif') != 0:
    # Non-zero returns indicate an error.
    print('ERROR running ruggedness_index')
```

If, like me, your data files tend to be burried deeply in layers of sub-directories, specifying complete file names as input parameters can be tedius. In this case, the best option is setting the working directory before calling tools:

```
from whitebox_tools import WhiteboxTools

wbt = WhiteboxTools()
wbt.work_dir = "/path/to/data/" # Sets the Whitebox working directory

# Because the working directory has been set, file arguments can be
# specified simply using file names, without paths.
wbt.d_inf_flow_accumulation("DEM.tif", "output.tif", log=True)
```

An advanced text editor, such as VS Code or Atom, can provide hints and autocompletion for available tool convenience methods and their parameters, including default values (Figure 1).

Sometimes it can be useful to print a complete list of available tools:

```
print(wbt.list_tools()) # List all tools in WhiteboxTools
```

The `list_tools()` method also takes an optional keywords list to search for tools:

```
# Lists tools with 'lidar' or 'LAS' in tool name or description.
print(wbt.list_tools(['lidar', 'LAS']))
```

To retrieve more detailed information for a specific tool, use the `tool_help()` method:

```
print(wbt.tool_help("elev_percentile"))
```

`tool_help()` prints tool details including a description, tool parameters (and their flags), and example usage at the command line prompt. The above statement prints this report:

```
ElevPercentile
Description:
```

Figure 1: Autocompletion in Atom text editor makes calling *WhiteboxTools* functions easier.

```
Calculates the elevation percentile raster from a DEM.
Toolbox: Geomorphometric Analysis
Parameters:

Flag                Description
-----------------   -----------
-i, --input, --dem Input raster DEM file.
-o, --output       Output raster file.
--filterx          Size of the filter kernel in the x-direction.
--filtery          Size of the filter kernel in the y-direction.
--sig_digits       Number of significant digits.

Example usage:
>>./whitebox_tools -r=ElevPercentile -v --wd="/path/to/data/" --dem=DEM.tif
>>-o=output.tif --filterx=25
```

A note on default parameter values

Each tool contains one or more parameters with default values. These will always be listed after any input parameters that do not have default values. You do not need to specify a parameter with a default value if you accept the default. That is, unless you intend to specify an input value different from the default, you may leave these parameters off of the function call. However, be mindful of the fact that Python assigns values to parameters based on order, unless parameter names are specified.

Consider the Hillshade tool as an example. The User Manual gives the following function definition for the tool:

```
hillshade(
dem,
output,
azimuth=315.0,
altitude=30.0,
zfactor=1.0,
callback=default_callback)
```

The `dem` and `output` parameters do not have default values and must be specified every time you call this function. Each of the remaining parameters have default values and can, optionally, be left off of calls to the `hillshade` function. As an example, say I want to accept the default values for all the parameters except `altitude`. I would then need to use the named-parameter form of the function call:

```
wbt.hillshade(
"DEM.tif",
"hillshade.tif",
altitude=20.0)
```

If I hadn't specified the parameter name for `altitude`, Python would have assumed that the value 20.0 should be assigned to the third parameter, `azimuth`.

5.2 Handling tool output

Tools will frequently print text to the standard output during their execution, including warnings, progress updates and other notifications. Sometimes, when users run many tools in complex workflows and in batch mode, these output messages can be undesirable. Most tools will have their outputs suppressed by setting the *verbose* mode to *False* as follows:

```
wbt.set_verbose_mode(False)
# Or, alternatively...
wbt.verbose = False
```

Alternatively, it may be helpful to capture the text output of a tool for custom processing. This is achieved by specifying a custom *callback* function to the tool's convenience method:

```
# This callback function suppresses printing progress updates,
# which always use the '%' character. The callback function
# approach is flexible and allows for any level of complex
# interaction with tool outputs.
def my_callback(value):
    if not "%" in value:
        print(value)

wbt.slope('DEM.tif', 'slope_raster.tif', callback=my_callback)
```

Callback functions can also serve as a means of cancelling operations:

```
def my_callback(value):
    if user_selected_cancel_btn: # Assumes a 'Cancel' button on a GUI
        print('Cancelling operation...')
        wbt.cancel_op = True
    else:
        print(value)

wbt.breach_depressions('DEM.tif', 'DEM_breached.tif', callback=my_callback)
```

5.3 Additional functions in *whitebox_tools.py*

The *whitebox_tools.py* script provides several other functions for interacting with the *WhiteboxTools* library, including:

```
# Print the WhiteboxTools help...a listing of available commands
print(wbt.help())

# Print the WhiteboxTools license
print(wbt.license())

# Print the WhiteboxTools version
print("Version information: {}".format(wbt.version()))

# Get the toolbox associated with a tool
tb = wbt.toolbox('lidar_info')

# Retrieve a JSON object of a tool's parameters.
tp = tool_parameters('raster_histogram')

# Opens a browser and navigates to a tool's source code in the
# WhiteboxTools GitHub repository
wbt.view_code('watershed')
```

For a working example of how to call functions and run tools from Python, see the *whitebox_example.py* Python script, which is distributed with the *WhiteboxTools* library.

5.4 An example WhiteboxTools Python project

In this section, we will create a Python project that utilizes the *WhiteboxTools* library to interpolate a LiDAR point-cloud, to process the resulting digital elevation model (DEM) to make it suitable for hydrological applications, and to perform a simple flow-accumulation operation. I suggest using an advanced coding text editor, such as *Visual Studio Code* or *Atom*, for this tutorial, but Python code can be written using any basic text editor.

Begin by creating a dedicated project directory called *FlowAccumExample* and copy *WhiteboxTools* binary file (i.e. the compressed file downloaded from the *Geomorphometry & Hydrogeomatics Research Group* website) into this folder. Using the decompression software on your computer, decompress (i.e. an operation sometimes called *unzipping*) the file into the newly created *FlowAccumExample* directory. You will find the compressed file contains a folder with contents similar to Figure 2.

The folder contains a number of files, including the *WhiteboxTools* executable file, the *whitebox_tools.py* python script, the WhiteboxTools Runner (*wb_runner.py*; see below), and this user manual. It is likely that the folder has a name that reflects the operating system and architecture that the binary file was compiled for (e.g. WhiteboxTools_darwin_amd64). Rename this directory to *WBT*. Also note, depending on your

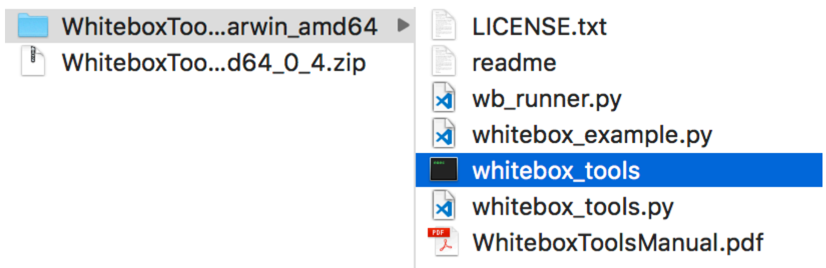

Figure 2: Folder contents of *WhiteboxTools* compressed download file

decompression software, it may be the case that the contents of the *WBT* folder itself contains a sub-directory that actually holds these files. If this is the case, be sure to move the contents of the sub-directory into the *WBT* parent directory.

Using your text editor, create a new Python script file, called *FlowAccumulation.py* within the *FlowAccumExample* directory. We will begin by importing the *WhiteboxTools* class from the *whitebox_tools.py* script contained within the *WBT* sub-directory. Unfortunately, Python's module system is only able to import classes and function definitions declared in external Python scripts *if these external files are contained somewhere on the Python path or in the directory containing the script file into which you are importing*. This is important because based on the project structure that we have established, the *whitebox_tools.py* script is actually contained within a sub-directory of the *FlowAccumExample* directory and is therefore not directly accessible, unless you have previously installed script on the Python path. Another, perhaps easier solution to this problem is to create a file named *__init__.py* (those are two leading and trailing underscore characters) within the *FlowAccumExample* directory. The presence of this empty file will make Python treat the *WBT* directory as containing packages, in this case, the *whitebox_tools* package. For more information, see the Python documentation on modules and packages.

At this stage, you should have a project directory structure like that of the Figure 3.

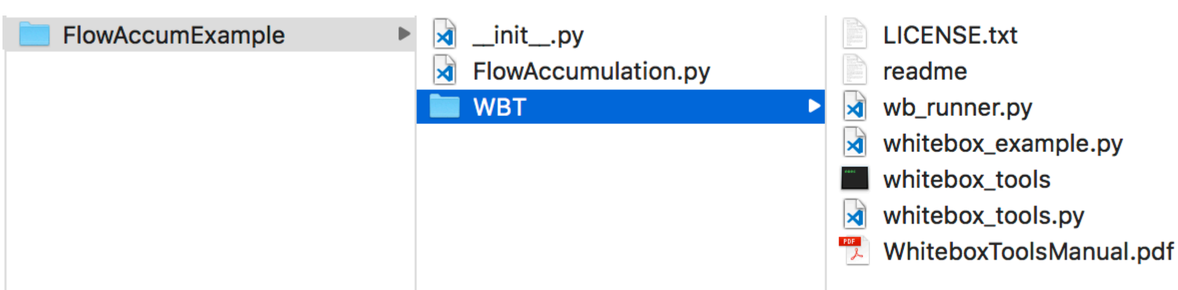

Figure 3: Example project set-up

Many operating systems will disallow the execution of files that are downloaded directly from the Internet. As such, it is possible that you will need to explicitly give the *whitebox_tools.exe* permission to execute on your computer (Note: here we are referring to the compiled *WhiteboxTools* binary file and not the similarly named Python script *whitebox_tools.py* also contained in the folder). The procedure for doing this depends on your specific operating system. On MacOS, for example, this is usually achieved using the 'Security & Privacy' tab under 'System Preferences'. To test whether *whitebox_tools.exe* has permission to run on your system, double-click the file. If the file is configured to execute, a command terminal will automatically open and the *WhiteboxTools* help documentation and a listing of the available tools will be printed. If this does not occur, you likely need to give the file permission to execute.

Using your text editor, you may now add the following lines to the *FlowAccumulation.py* file.

```
from WBT.whitebox_tools import WhiteboxTools

wbt = WhiteboxTools()
```

In the import statement, `WBT` is a reference to the package folder containing the *WhiteboxTools* files; `whitebox_tools` is a reference to the *whitebox_tools.py* script contained with this package folder; and `WhiteboxTools` is a reference to the *WhiteboxTools* class contained within this script file. Please note that if you named your directory containing the *WhiteboxTools* files something other than *WBT*, you would need to alter the import statement accordingly.

Visit the *Geomorphometry and Hydrogeomatics Research Group* website and download the St. Elis Mountains and Gulf of Alaska sample data set (*StElisAk.las*) from the *WhiteboxTools* section of the site. This file contains a LiDAR point cloud that has been previously filtered to remove points associated with non-ground returns, mainly trees (Figure 4). Create a sub-directory within the project folder called 'data' and copy *StElisAk.las* into the folder.

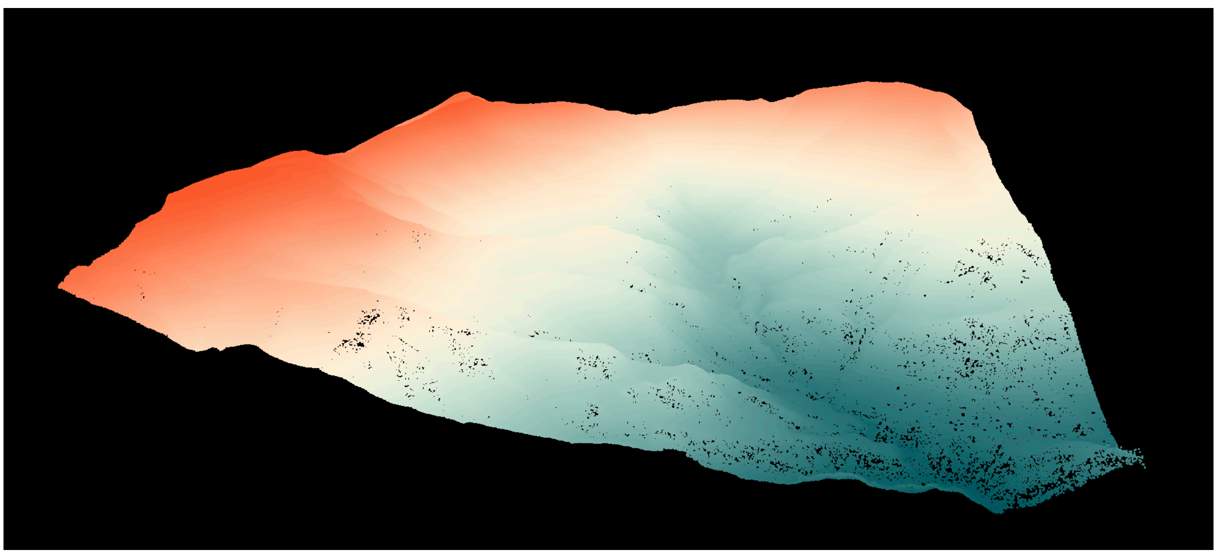

Figure 4: St. Elis Mountains LiDAR point cloud, visualized using the plas.io software

Now we can complete our flow accumulation analysis with the following code:

```
import os
from WBT.whitebox_tools import WhiteboxTools

wbt = WhiteboxTools()

# Set the working directory, i.e. the folder containing the data,
# to the 'data' sub-directory.
```

```
wbt.work_dir = os.path.dirname(os.path.abspath(__file__)) + "/data/"

# When you're running mulitple tools, the outputs can be a tad
# chatty. In this case, you may want to suppress the output by
# setting the verbose mode to False.
# wbt.verbose = False

# Interpolate the LiDAR data using an inverse-distance weighting
# (IDW) scheme.
print("Interpolating DEM...")
wbt.lidar_idw_interpolation(
i="StElisAk.las",
output="raw_dem.tif",
parameter="elevation",
returns="last",
resolution=1.0,
weight=1.0,
radius=2.5
)

# The resulting DEM will contain NoData gaps. We need to fill
# these in by interpolating across the gap.
print("Filling missing data...")
wbt.fill_missing_data(
i="raw_dem.tif",
output="dem_nodata_filled.tif",
filter=11
)

# This DEM will contain grid cells that have no lower neighbours.
# This condition is unsuited for flow-path modelling applications
# because these operations assume that each interior cell in the
# DEM has at least one downslope neighour. We'll use an operation
# called depression breaching to 'fix' the elevations within the
# DEM to enforce continuous flow.
print("Performing flow enforcement...")
wbt.breach_depressions(
dem="dem_nodata_filled.tif",
output="dem_hydro_enforced.tif"
)

# Lastly, perform the flow accumulation operation using the
# D-infinity flow algorithm.
print("Performing flow accumulation...")
```

```python
wbt.d_inf_flow_accumulation(
dem="dem_hydro_enforced.tif",
output="flow_accum.tif",
log=True
)

print("Complete!")
```

To run the above script, open a terminal (command prompt), *cd* to the script containing folder, and run the following command:

```
>>python FlowAccumulation.py
```

If Python 3 is not your default Python version, substitute `python3` for `python` in the above command line. The final D-infinity flow accumulation raster can be displayed in any GIS software of choice and should look similar to Figure 5.

6. WhiteboxTools Runner

There is a Python script contained within the *WhiteboxTools* directory called '*wb_runner.py*'. This script is intended to provide a very basic user-interface, *WhiteboxTools Runner*, for running the tools contained within the *WhiteboxTools* library. The user-interface uses Python's TkInter GUI library and is cross-platform. The user interface is currently experimental and is under heavy testing. Please report any issues that you experience in using it.

The *WhiteboxTools Runner* does not rely on the *Whitebox GAT* user interface at all and can therefore be used indepedent of the larger project. The script must be run from a directory that also contains the '*whitebox_tools.py*' Python script and the '*whitebox_tools*' executable file. There are plans to link tool help documentation in *WhiteboxTools Runner* and to incorporate toolbox information, rather than one large listing of available tools.

6. Available Tools

Eventually most of *Whitebox GAT's* approximately 400 tools will be ported to *WhiteboxTools*, although this is an immense task. Support for vector data (Shapefile/GeoJSON) reading/writing and a topological analysis library (like the Java Topology Suite) will need to be added in order to port the tools involving vector spatial data. Opportunities to parallelize algorithms will be sought during porting. All new plugin tools will be added to *Whitebox GAT* using this library of functions.

The library currently contains the following 293 tools, which are each grouped based on their main function into one of the following categories: *Data Tools*, *Geomorphometric Analysis* (i.e. digital terrain analysis), *GIS Analysis*, *Hydrological Analysis*, *Image Analysis*, *LiDAR Analysis*, *Mathematical and Statistical Analysis*, and

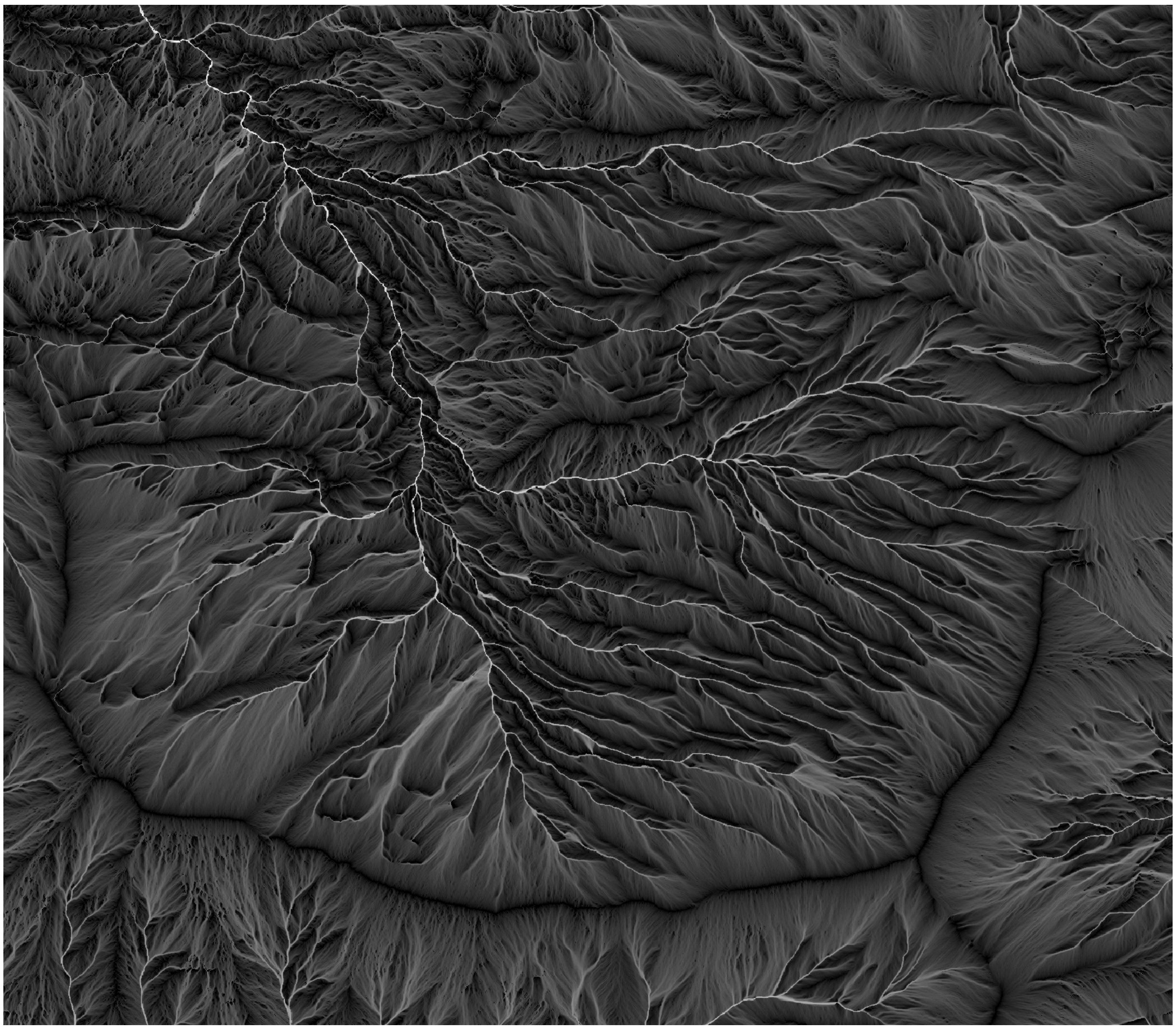

Figure 5: Output of the flow accumulation script for the St. Elis Mountains data set.

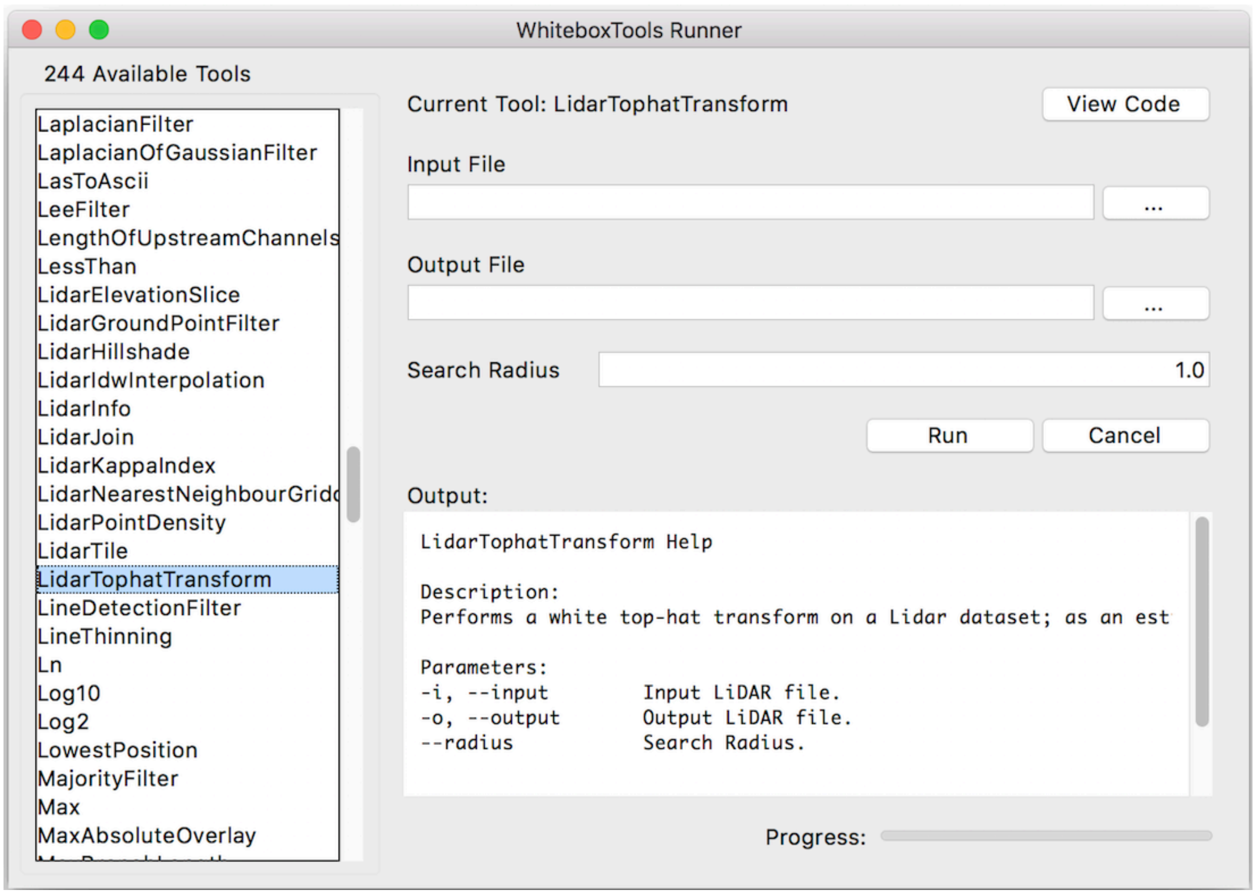

Figure 6: The *WhiteboxTools Runner* user-interface

Stream Network Analysis. To retrieve detailed information about a tool's input arguments and example usage, either use the `toolhelp` command from the terminal, or the `tool_help('tool_name')` function from the *whitebox_tools.py* script. The following is a complete listing of available tools, with brief descriptions, tool parameter, and example usage.

7.1 Data Tools

7.1.1 ConvertNodataToZero

Converts nodata values in a raster to zero.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
convert_nodata_to_zero(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ConvertNodataToZero -v ^
--wd="/path/to/data/" --input=in.tif -o=NewRaster.tif
```

7.1.2 ConvertRasterFormat

Converts raster data from one format to another.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
convert_raster_format(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ConvertRasterFormat -v ^
--wd="/path/to/data/" --input=DEM.tif -o=output.tif
```

7.1.3 NewRasterFromBase

Creates a new raster using a base image.

Parameters:

Flag	Description
-i, --base	Input base raster file
-o, --output	Output raster file
--value	Constant value to fill raster with; either 'nodata' or numeric value
--data_type	Output raster data type; options include 'double' (64-bit), 'float' (32-bit), and 'integer' (signed 16-bit) (default is 'float')

Python function:

```
new_raster_from_base(
    base,
    output,
    value="nodata",
    data_type="float",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=NewRasterFromBase -v ^
--wd="/path/to/data/" --base=base.tif -o=NewRaster.tif ^
--value=0.0 --data_type=integer
>>./whitebox_tools ^
-r=NewRasterFromBase -v --wd="/path/to/data/" --base=base.tif ^
-o=NewRaster.tif --value=nodata
```

7.1.4 PrintGeoTiffTags

Prints the tags within a GeoTIFF.

Parameters:

Flag	Description
-i, --input	Input GeoTIFF file

Python function:

```
print_geo_tiff_tags(
    i,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PrintGeoTiffTags -v ^
--wd="/path/to/data/" --input=DEM.tiff
```

7.1.5 SetNodataValue

Assign a specified value in an input image to the NoData value.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--back_value	Background value to set to nodata

Python function:

```
set_nodata_value(
    i,
    output,
    back_value=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SetNodataValue -v --wd="/path/to/data/" ^
-i=in.tif -o=newRaster.tif --back_value=1.0
```

7.1.6 VectorLinesToRaster

Converts a vector containing polylines into a raster.

Parameters:

Flag	Description
-i, --input	Input vector lines file
--field	Input field name in attribute table
-o, --output	Output raster file
--nodata	Background value to set to NoData. Without this flag, it will be set to 0.0
--cell_size	Optionally specified cell size of output raster. Not used when base raster is specified
--base	Optionally specified input base raster file. Not used when a cell size is specified

Python function:

```
vector_lines_to_raster(
    i,
    output,
    field="FID",
    nodata=True,
    cell_size=None,
    base=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=VectorLinesToRaster -v ^
--wd="/path/to/data/" -i=lines.shp --field=ELEV -o=output.tif ^
--nodata --cell_size=10.0
        >>./whitebox_tools ^
-r=VectorLinesToRaster -v --wd="/path/to/data/" -i=lines.shp ^
--field=FID -o=output.tif --base=existing_raster.tif
```

7.1.7 VectorPointsToRaster

Converts a vector containing points into a raster.

Parameters:

Flag	Description
-i, --input	Input vector Points file

Flag	Description
--field	Input field name in attribute table
-o, --output	Output raster file
--assign	Assignment operation, where multiple points are in the same grid cell; options include 'first', 'last' (default), 'min', 'max', 'sum'
--nodata	Background value to set to NoData. Without this flag, it will be set to 0.0
--cell_size	Optionally specified cell size of output raster. Not used when base raster is specified
--base	Optionally specified input base raster file. Not used when a cell size is specified

Python function:

```
vector_points_to_raster(
    i,
    output,
    field="FID",
    assign="last",
    nodata=True,
    cell_size=None,
    base=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=VectorPointsToRaster -v ^
--wd="/path/to/data/" -i=points.shp --field=ELEV -o=output.tif ^
--assign=min --nodata ^
--cell_size=10.0
        >>./whitebox_tools ^
-r=VectorPointsToRaster -v --wd="/path/to/data/" -i=points.shp ^
--field=FID -o=output.tif --assign=last ^
--base=existing_raster.tif
```

7.1.8 VectorPolygonsToRaster

Converts a vector containing polygons into a raster.

Parameters:

Flag	Description
-i, --input	Input vector polygons file
--field	Input field name in attribute table

Flag	Description
-o, --output	Output raster file
--nodata	Background value to set to NoData. Without this flag, it will be set to 0.0
--cell_size	Optionally specified cell size of output raster. Not used when base raster is specified
--base	Optionally specified input base raster file. Not used when a cell size is specified

Python function:

```
vector_polygons_to_raster(
    i,
    output,
    field="FID",
    nodata=True,
    cell_size=None,
    base=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=VectorPolygonsToRaster -v ^
--wd="/path/to/data/" -i=lakes.shp --field=ELEV -o=output.tif ^
--nodata --cell_size=10.0
        >>./whitebox_tools ^
-r=VectorPolygonsToRaster -v --wd="/path/to/data/" ^
-i=lakes.shp --field=ELEV -o=output.tif ^
--base=existing_raster.tif
```

7.2 GIS Analysis

7.2.1 AggregateRaster

Aggregates a raster to a lower resolution.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--agg_factor	Aggregation factor, in pixels
--type	Statistic used to fill output pixels

Python function:

```
aggregate_raster(
    i,
    output,
    agg_factor=2,
    type="mean",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AggregateRaster -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif ^
--output_text
```

7.2.2 Centroid

Calculates the centroid, or average location, of raster polygon objects.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--text_output	Optional text output

Python function:

```
centroid(
    i,
    output,
    text_output=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Centroid -v --wd="/path/to/data/" ^
-i=polygons.tif -o=output.tif
>>./whitebox_tools -r=Centroid ^
-v --wd="/path/to/data/" -i=polygons.tif -o=output.tif ^
--text_output
```

7.2.3 Clump

Groups cells that form physically discrete areas, assigning them unique identifiers.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--diag	Flag indicating whether diagonal connections should be considered
--zero_back	Flag indicating whether zero values should be treated as a background

Python function:

```
clump(
    i,
    output,
    diag=True,
    zero_back=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Clump -v --wd="/path/to/data/" ^
-i=input.tif -o=output.tif --diag
```

7.2.4 CreatePlane

Creates a raster image based on the equation for a simple plane.

Parameters:

Flag	Description
--base	Input base raster file
-o, --output	Output raster file
--gradient	Slope gradient in degrees (-85.0 to 85.0)
--aspect	Aspect (direction) in degrees clockwise from north (0.0-360.0)
--constant	Constant value

Python function:

```
create_plane(
    base,
    output,
```

```
    gradient=15.0,
    aspect=90.0,
    constant=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CreatePlane -v --wd="/path/to/data/" ^
--base=base.tif -o=NewRaster.tif --gradient=15.0 ^
--aspect=315.0
```

7.2.5 RasterCellAssignment

Assign row or column number to cells.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
-a, --assign	Which variable would you like to assign to grid cells? Options include 'column', 'row', 'x', and 'y'

Python function:

```
raster_cell_assignment(
    i,
    output,
    assign="column",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RasterCellAssignment -v ^
--wd="/path/to/data/" -i='input.tif' -o=output.tif ^
--assign='column'
```

7.2.6 Reclass

Reclassifies the values in a raster image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--reclass_vals	Reclassification triplet values (new value; from value; to less than), e.g. '0.0;0.0;1.0;1.0;1.0;2.0'
--assign_mode	Optional Boolean flag indicating whether to operate in assign mode, reclass_vals values are interpreted as new value; old value pairs

Python function:

```
reclass(
    i,
    output,
    reclass_vals,
    assign_mode=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Reclass -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif ^
--reclass_vals='0.0;0.0;1.0;1.0;1.0;2.0'
>>./whitebox_tools ^
-r=Reclass -v --wd="/path/to/data/" -i='input.tif' ^
-o=output.tif --reclass_vals='10;1;20;2;30;3;40;4' ^
--assign_mode
```

7.2.7 ReclassEqualInterval

Reclassifies the values in a raster image based on equal-ranges.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--interval	Class interval size
--start_val	Optional starting value (default is input minimum value)
--end_val	Optional ending value (default is input maximum value)

Python function:

```
reclass_equal_interval(
    i,
    output,
    interval=10.0,
    start_val=None,
    end_val=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ReclassEqualInterval -v ^
--wd="/path/to/data/" -i='input.tif' -o=output.tif ^
--interval=10.0 --start_val=0.0
```

7.2.8 ReclassFromFile

Reclassifies the values in a raster image using reclass ranges in a text file.

Parameters:

Flag	Description
-i, --input	Input raster file
--reclass_file	Input text file containing reclass ranges
-o, --output	Output raster file

Python function:

```
reclass_from_file(
    i,
    reclass_file,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ReclassFromFile -v ^
--wd="/path/to/data/" -i='input.tif' ^
--reclass_file='reclass.txt' -o=output.tif
```

7.3 GIS Analysis => Distance Tools

7.3.1 BufferRaster

Maps a distance-based buffer around each non-background (non-zero/non-nodata) grid cell in an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--size	Buffer size
--gridcells	Optional flag to indicate that the 'size' threshold should be measured in grid cells instead of the default map units

Python function:

```
buffer_raster(
    i,
    output,
    size,
    gridcells=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=BufferRaster -v --wd="/path/to/data/" ^
-i=DEM.tif -o=output.tif
```

7.3.2 CostAllocation

Identifies the source cell to which each grid cell is connected by a least-cost pathway in a cost-distance analysis.

Parameters:

Flag	Description
--source	Input source raster file
--backlink	Input backlink raster file generated by the cost-distance tool
-o, --output	Output raster file

Python function:

```
cost_allocation(
    source,
    backlink,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CostAllocation -v --wd="/path/to/data/" ^
--source='source.tif' --backlink='backlink.tif' ^
-o='output.tif'
```

7.3.3 CostDistance

Performs cost-distance accumulation on a cost surface and a group of source cells.

Parameters:

Flag	Description
--source	Input source raster file
--cost	Input cost (friction) raster file
--out_accum	Output cost accumulation raster file
--out_backlink	Output backlink raster file

Python function:

```
cost_distance(
    source,
    cost,
    out_accum,
    out_backlink,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CostDistance -v --wd="/path/to/data/" ^
--source=src.tif --cost=cost.tif --out_accum=accum.tif ^
--out_backlink=backlink.tif
```

7.3.4 CostPathway

Performs cost-distance pathway analysis using a series of destination grid cells.

Parameters:

Flag	Description
--destination	Input destination raster file
--backlink	Input backlink raster file generated by the cost-distance tool
-o, --output	Output cost pathway raster file
--zero_background	Flag indicating whether zero values should be treated as a background

Python function:

```
cost_pathway(
    destination,
    backlink,
    output,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CostPathway -v --wd="/path/to/data/" ^
--destination=dst.tif --backlink=backlink.tif ^
--output=cost_path.tif
```

7.3.5 EuclideanAllocation

Assigns grid cells in the output raster the value of the nearest target cell in the input image, measured by the Shih and Wu (2004) Euclidean distance transform.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
euclidean_allocation(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=EuclideanAllocation -v ^
```

```
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.3.6 EuclideanDistance

Calculates the Shih and Wu (2004) Euclidean distance transform.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
euclidean_distance(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=EuclideanDistance -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.4 GIS Analysis => Overlay Tools

7.4.1 AverageOverlay

Calculates the average for each grid cell from a group of raster images.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file

Python function:

```
average_overlay(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AverageOverlay -v --wd='/path/to/data/' ^
-i='image1.dep;image2.dep;image3.tif' -o=output.tif
```

7.4.2 CountIf

Counts the number of occurrences of a specified value in a cell-stack of rasters.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file
--value	Search value (e.g. countif value = 5.0)

Python function:

```
count_if(
    inputs,
    output,
    value,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CountIf -v --wd='/path/to/data/' ^
-i='image1.dep;image2.dep;image3.tif' -o=output.tif ^
--value=5.0
```

7.4.3 ErasePolygonFromRaster

Erases (cuts out) a vector polygon from a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
--polygons	Input vector polygons file
-o, --output	Output raster file

Python function:

```
erase_polygon_from_raster(
    i,
    polygons,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ErasePolygonFromRaster -v ^
--wd="/path/to/data/" -i='DEM.tif' --polygons='lakes.shp' ^
-o='output.tif'
```

7.4.4 HighestPosition

Identifies the stack position of the maximum value within a raster stack on a cell-by-cell basis.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file

Python function:

```
highest_position(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HighestPosition -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
-o=output.tif
```

7.4.5 LowestPosition

Identifies the stack position of the minimum value within a raster stack on a cell-by-cell basis.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file

Python function:

```
lowest_position(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LowestPosition -v --wd='/path/to/data/' ^
-i='image1.tif;image2.tif;image3.tif' -o=output.tif
```

7.4.6 MaxAbsoluteOverlay

Evaluates the maximum absolute value for each grid cell from a stack of input rasters.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file

Python function:

```
max_absolute_overlay(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxAbsoluteOverlay -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
-o=output.tif
```

7.4.7 MaxOverlay

Evaluates the maximum value for each grid cell from a stack of input rasters.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file

Python function:

```
max_overlay(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxOverlay -v --wd='/path/to/data/' ^
-i='image1.tif;image2.tif;image3.tif' -o=output.tif
```

7.4.8 MinAbsoluteOverlay

Evaluates the minimum absolute value for each grid cell from a stack of input rasters.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file

Python function:

```
min_absolute_overlay(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MinAbsoluteOverlay -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
-o=output.tif
```

7.4.9 MinOverlay

Evaluates the minimum value for each grid cell from a stack of input rasters.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file

Python function:

```
min_overlay(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MinOverlay -v --wd='/path/to/data/' ^
-i='image1.tif;image2.tif;image3.tif' -o=output.tif
```

7.4.10 PercentEqualTo

Calculates the percentage of a raster stack that have cell values equal to an input on a cell-by-cell basis.

Parameters:

Flag	Description
-i, --inputs	Input raster files
--comparison	Input comparison raster file
-o, --output	Output raster file

Python function:

```
percent_equal_to(
    inputs,
    comparison,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PercentEqualTo -v --wd='/path/to/data/' ^
```

```
-i='image1.tif;image2.tif;image3.tif' --comparison='comp.tif' ^
-o='output.tif'
```

7.4.11 PercentGreaterThan

Calculates the percentage of a raster stack that have cell values greather than an input on a cell-by-cell basis.

Parameters:

Flag	Description
-i, --inputs	Input raster files
--comparison	Input comparison raster file
-o, --output	Output raster file

Python function:

```
percent_greater_than(
    inputs,
    comparison,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PercentGreaterThan -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
--comparison='comp.tif' -o='output.tif'
```

7.4.12 PercentLessThan

Calculates the percentage of a raster stack that have cell values less than an input on a cell-by-cell basis.

Parameters:

Flag	Description
-i, --inputs	Input raster files
--comparison	Input comparison raster file
-o, --output	Output raster file

Python function:

```
percent_less_than(
    inputs,
    comparison,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PercentLessThan -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
--comparison='comp.tif' -o='output.tif'
```

7.4.13 PickFromList

Outputs the value from a raster stack specified by a position raster.

Parameters:

Flag	Description
-i, --inputs	Input raster files
--pos_input	Input position raster file
-o, --output	Output raster file

Python function:

```
pick_from_list(
    inputs,
    pos_input,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PickFromList -v --wd='/path/to/data/' ^
--pos_input=position.tif -i='image1.tif;image2.tif;image3.tif' ^
-o=output.tif
```

7.4.14 WeightedSum

Performs a weighted-sum overlay on multiple input raster images.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file
-w, --weights	Weight values, contained in quotes and separated by commas or semicolons

Python function:

```
weighted_sum(
    inputs,
    output,
    weights,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=WeightedSum -v --wd='/path/to/data/' ^
-i='image1.tif;image2.tif;image3.tif' -o=output.tif ^
--weights='0.3;0.2;0.5'
```

7.5 GIS Analysis => Patch Shape Tools

7.5.1 EdgeProportion

Calculate the proportion of cells in a raster polygon that are edge cells.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--output_text	flag indicating whether a text report should also be output

Python function:

```
edge_proportion(
    i,
    output,
    output_text=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=EdgeProportion -v --wd="/path/to/data/" ^
```

```
-i=input.tif -o=output.tif --output_text
```

7.5.2 FindPatchOrClassEdgeCells

Finds all cells located on the edge of patch or class features.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
find_patch_or_class_edge_cells(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FindPatchOrClassEdgeCells -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif
```

7.5.3 RadiusOfGyration

Calculates the distance of cells from their polygon's centroid.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--text_output	Optional text output

Python function:

```
radius_of_gyration(
    i,
    output,
    text_output=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RadiusOfGyration -v ^
--wd="/path/to/data/" -i=polygons.tif -o=output.tif ^
--text_output
```

7.6 Geomorphometric Analysis

7.6.1 Aspect

Calculates an aspect raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
aspect(
    dem,
    output,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Aspect -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.6.2 DevFromMeanElev

Calculates deviation from mean elevation.

Parameters:

Flag	Description
-i, --input, --dem	Input raster DEM file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction

Flag	Description
--filtery	Size of the filter kernel in the y-direction

Python function:

```
dev_from_mean_elev(
    dem,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DevFromMeanElev -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--filter=25
```

7.6.3 DiffFromMeanElev

Calculates difference from mean elevation (equivalent to a high-pass filter).

Parameters:

Flag	Description
-i, --input, --dem	Input raster DEM file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
diff_from_mean_elev(
    dem,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DiffFromMeanElev -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--filter=25
```

7.6.4 DirectionalRelief

Calculates relief for cells in an input DEM for a specified direction.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--azimuth	Wind azimuth in degrees
--max_dist	Optional maximum search distance (unspecified if none; in xy units)

Python function:

```
directional_relief(
    dem,
    output,
    azimuth=0.0,
    max_dist=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DirectionalRelief -v ^
--wd="/path/to/data/" -i='input.tif' -o=output.tif ^
--azimuth=315.0
```

7.6.5 DownslopeIndex

Calculates the Hjerdt et al. (2004) downslope index.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--drop	Vertical drop value (default is 2.0)
--out_type	Output type, options include 'tangent', 'degrees', 'radians', 'distance' (default is 'tangent')

Python function:

```
downslope_index(
    dem,
    output,
    drop=2.0,
    out_type="tangent",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DownslopeIndex -v --wd="/path/to/data/" ^
--dem=pointer.tif -o=dsi.tif --drop=5.0 --out_type=distance
```

7.6.6 ElevAbovePit

Calculate the elevation of each grid cell above the nearest downstream pit cell or grid edge cell.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
elev_above_pit(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ElevAbovePit -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.6.7 ElevPercentile

Calculates the elevation percentile raster from a DEM.

Parameters:

Flag	Description
-i, --input, --dem	Input raster DEM file
-o, --output	Output raster file

Flag	Description
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction
--sig_digits	Number of significant digits

Python function:

```
elev_percentile(
    dem,
    output,
    filterx=11,
    filtery=11,
    sig_digits=2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ElevPercentile -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif --filter=25
```

7.6.8 ElevRelativeToMinMax

Calculates the elevation of a location relative to the minimum and maximum elevations in a DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
elev_relative_to_min_max(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ElevRelativeToMinMax -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.6.9 ElevRelativeToWatershedMinMax

Calculates the elevation of a location relative to the minimum and maximum elevations in a watershed.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--watersheds	Input raster watersheds file
-o, --output	Output raster file

Python function:

```
elev_relative_to_watershed_min_max(
    dem,
    watersheds,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ElevRelativeToWatershedMinMax -v ^
--wd="/path/to/data/" --dem=DEM.tif --watersheds=watershed.tif ^
-o=output.tif
```

7.6.10 FeaturePreservingDenoise

Reduces short-scale variation in an input DEM using a modified Sun et al. (2007) algorithm.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--filter	Size of the filter kernel
--norm_diff	Maximum difference in normal vectors, in degrees
--num_iter	Number of iterations
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
feature_preserving_denoise(
    dem,
    output,
```

```
    filter=11,
    norm_diff=15.0,
    num_iter=5,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FeaturePreservingDenoise -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.6.11 FetchAnalysis

Performs an analysis of fetch or upwind distance to an obstacle.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--azimuth	Wind azimuth in degrees in degrees
--hgt_inc	Height increment value

Python function:

```
fetch_analysis(
    dem,
    output,
    azimuth=0.0,
    hgt_inc=0.05,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FetchAnalysis -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif --azimuth=315.0
```

7.6.12 FillMissingData

Fills nodata holes in a DEM.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filter	Filter size (cells)

Python function:

```
fill_missing_data(
    i,
    output,
    filter=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FillMissingData -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif --filter=25
```

7.6.13 FindRidges

Identifies potential ridge and peak grid cells.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--line_thin	Optional flag indicating whether post-processing line-thinning should be performed

Python function:

```
find_ridges(
    dem,
    output,
    line_thin=True,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FindRidges -v --wd="/path/to/data/" ^
--dem=pointer.tif -o=out.tif --line_thin
```

7.6.14 Hillshade

Calculates a hillshade raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--azimuth	Illumination source azimuth in degrees
--altitude	Illumination source altitude in degrees
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
hillshade(
    dem,
    output,
    azimuth=315.0,
    altitude=30.0,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Hillshade -v --wd="/path/to/data/" ^
-i=DEM.tif -o=output.tif --azimuth=315.0 --altitude=30.0
```

7.6.15 HorizonAngle

Calculates horizon angle (maximum upwind slope) for each grid cell in an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--azimuth	Wind azimuth in degrees
--max_dist	Optional maximum search distance (unspecified if none; in xy units)

Python function:

```
horizon_angle(
    dem,
    output,
    azimuth=0.0,
    max_dist=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HorizonAngle -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif --azimuth=315.0
```

7.6.16 HypsometricAnalysis

Calculates a hypsometric curve for one or more DEMs.

Parameters:

Flag	Description
-i, --inputs	Input DEM files
--watershed	Input watershed files (optional)
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
hypsometric_analysis(
    inputs,
    output,
    watershed=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HypsometricAnalysis -v ^
--wd="/path/to/data/" -i="DEM1.tif;DEM2.tif" ^
--watershed="ws1.tif;ws2.tif" -o=outfile.html
```

7.6.17 MaxAnisotropyDev

Calculates the maximum anisotropy (directionality) in elevation deviation over a range of spatial scales.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--out_mag	Output raster DEVmax magnitude file
--out_scale	Output raster DEVmax scale file
--min_scale	Minimum search neighbourhood radius in grid cells
--max_scale	Maximum search neighbourhood radius in grid cells
--step	Step size as any positive non-zero integer

Python function:

```
max_anisotropy_dev(
    dem,
    out_mag,
    out_scale,
    max_scale,
    min_scale=3,
    step=2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxAnisotropyDev -v ^
--wd="/path/to/data/" --dem=DEM.tif --out_mag=DEVmax_mag.tif ^
--out_scale=DEVmax_scale.tif --min_scale=1 --max_scale=1000 ^
--step=5
```

7.6.18 MaxAnisotropyDevSignature

Calculates the anisotropy in deviation from mean for points over a range of spatial scales.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--points	Input vector points file
-o, --output	Output HTML file
--min_scale	Minimum search neighbourhood radius in grid cells
--max_scale	Maximum search neighbourhood radius in grid cells
--step	Step size as any positive non-zero integer

Python function:

```
max_anisotropy_dev_signature(
    dem,
    points,
    output,
    max_scale,
    min_scale=1,
    step=1,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxAnisotropyDevSignature -v ^
--wd="/path/to/data/" --dem=DEM.tif --points=sites.shp ^
--output=roughness.html --min_scale=1 --max_scale=1000 ^
--step=5
```

7.6.19 MaxBranchLength

Lindsay and Seibert's (2013) branch length index is used to map drainage divides or ridge lines.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--log	Optional flag to request the output be log-transformed

Python function:

```
max_branch_length(
    dem,
    output,
    log=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxBranchLength -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.6.20 MaxDownslopeElevChange

Calculates the maximum downslope change in elevation between a grid cell and its eight downslope neighbors.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
max_downslope_elev_change(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxDownslopeElevChange -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=out.tif
```

7.6.21 MaxElevDevSignature

Calculates the maximum elevation deviation over a range of spatial scales and for a set of points.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--points	Input vector points file
-o, --output	Output HTML file
--min_scale	Minimum search neighbourhood radius in grid cells
--max_scale	Maximum search neighbourhood radius in grid cells
--step	Step size as any positive non-zero integer

Python function:

```
max_elev_dev_signature(
    dem,
    points,
    output,
    min_scale,
```

```
    max_scale,
    step=10,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxElevDevSignature -v ^
--wd="/path/to/data/" --dem=DEM.tif --points=sites.tif ^
--output=topo_position.html --min_scale=1 --max_scale=1000 ^
--step=5
```

7.6.22 MaxElevationDeviation

Calculates the maximum elevation deviation over a range of spatial scales.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--out_mag	Output raster DEVmax magnitude file
--out_scale	Output raster DEVmax scale file
--min_scale	Minimum search neighbourhood radius in grid cells
--max_scale	Maximum search neighbourhood radius in grid cells
--step	Step size as any positive non-zero integer

Python function:

```
max_elevation_deviation(
    dem,
    out_mag,
    out_scale,
    min_scale,
    max_scale,
    step=10,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxElevationDeviation -v ^
--wd="/path/to/data/" --dem=DEM.tif --out_mag=DEVmax_mag.tif ^
--out_scale=DEVmax_scale.tif --min_scale=1 --max_scale=1000 ^
--step=5
```

7.6.23 MinDownslopeElevChange

Calculates the minimum downslope change in elevation between a grid cell and its eight downslope neighbors.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
min_downslope_elev_change(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MinDownslopeElevChange -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=out.tif
```

7.6.24 MultiscaleRoughness

Calculates surface roughness over a range of spatial scales.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--out_mag	Output raster roughness magnitude file
--out_scale	Output raster roughness scale file
--min_scale	Minimum search neighbourhood radius in grid cells
--max_scale	Maximum search neighbourhood radius in grid cells
--step	Step size as any positive non-zero integer

Python function:

```
multiscale_roughness(
    dem,
    out_mag,
    out_scale,
    max_scale,
```

```
    min_scale=1,
    step=1,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MultiscaleRoughness -v ^
--wd="/path/to/data/" --dem=DEM.tif --out_mag=roughness_mag.tif ^
--out_scale=roughness_scale.tif --min_scale=1 --max_scale=1000 ^
--step=5
```

7.6.25 MultiscaleRoughnessSignature

Calculates the surface roughness for points over a range of spatial scales.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--points	Input vector points file
-o, --output	Output HTML file
--min_scale	Minimum search neighbourhood radius in grid cells
--max_scale	Maximum search neighbourhood radius in grid cells
--step	Step size as any positive non-zero integer

Python function:

```
multiscale_roughness_signature(
    dem,
    points,
    output,
    max_scale,
    min_scale=1,
    step=1,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MultiscaleRoughnessSignature -v ^
--wd="/path/to/data/" --dem=DEM.tif --points=sites.shp ^
--output=roughness.html --min_scale=1 --max_scale=1000 ^
--step=5
```

7.6.26 MultiscaleTopographicPositionImage

Creates a multiscale topographic position image from three DEVmax rasters of differing spatial scale ranges.

Parameters:

Flag	Description
--local	Input local-scale topographic position (DEVmax) raster file
--meso	Input meso-scale topographic position (DEVmax) raster file
--broad	Input broad-scale topographic position (DEVmax) raster file
-o, --output	Output raster file
--lightness	Image lightness value (default is 1.2)

Python function:

```
multiscale_topographic_position_image(
    local,
    meso,
    broad,
    output,
    lightness=1.2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MultiscaleTopographicPositionImage -v ^
--wd="/path/to/data/" --local=DEV_local.tif --meso=DEV_meso.tif ^
--broad=DEV_broad.tif -o=output.tif --lightness=1.5
```

7.6.27 NumDownslopeNeighbours

Calculates the number of downslope neighbours to each grid cell in a DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
num_downslope_neighbours(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=NumDownslopeNeighbours -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.6.28 NumUpslopeNeighbours

Calculates the number of upslope neighbours to each grid cell in a DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
num_upslope_neighbours(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=NumUpslopeNeighbours -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.6.29 PennockLandformClass

Classifies hillslope zones based on slope, profile curvature, and plan curvature.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--slope	Slope threshold value, in degrees (default is 3.0)
--prof	Profile curvature threshold value (default is 0.1)

Flag	Description
--plan	Plan curvature threshold value (default is 0.0)
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
pennock_landform_class(
    dem,
    output,
    slope=3.0,
    prof=0.1,
    plan=0.0,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PennockLandformClass -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif --slope=3.0 ^
--prof=0.1 --plan=0.0
```

7.6.30 PercentElevRange

Calculates percent of elevation range from a DEM.

Parameters:

Flag	Description
-i, --input, --dem	Input raster DEM file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
percent_elev_range(
    dem,
    output,
    filterx=3,
    filtery=3,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PercentElevRange -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif --filter=25
```

7.6.31 PlanCurvature

Calculates a plan (contour) curvature raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
plan_curvature(
    dem,
    output,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PlanCurvature -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.6.32 Profile

Plots profiles from digital surface models.

Parameters:

Flag	Description
--lines	Input vector line file
--surface	Input raster surface file
-o, --output	Output HTML file

Python function:

```
profile(
    lines,
```

```
    surface,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Profile -v --wd="/path/to/data/" ^
--lines=profile.shp --surface=dem.tif -o=profile.html
```

7.6.33 ProfileCurvature

Calculates a profile curvature raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
profile_curvature(
    dem,
    output,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ProfileCurvature -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.6.34 RelativeAspect

Calculates relative aspect (relative to a user-specified direction) from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--azimuth	Illumination source azimuth

Flag	Description
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
relative_aspect(
    dem,
    output,
    azimuth=0.0,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RelativeAspect -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif --azimuth=180.0
```

7.6.35 RelativeStreamPowerIndex

Calculates the relative stream power index.

Parameters:

Flag	Description
--sca	Input raster specific contributing area (SCA) file
--slope	Input raster slope file
-o, --output	Output raster file
--exponent	SCA exponent value

Python function:

```
relative_stream_power_index(
    sca,
    slope,
    output,
    exponent=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RelativeStreamPowerIndex -v ^
--wd="/path/to/data/" --sca='flow_accum.tif' ^
--slope='slope.tif' -o=output.tif --exponent=1.1
```

7.6.36 RelativeTopographicPosition

Calculates the relative topographic position index from a DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
relative_topographic_position(
    dem,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RelativeTopographicPosition -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--filter=25
```

7.6.37 RemoveOffTerrainObjects

Removes off-terrain objects from a raster digital elevation model (DEM).

Parameters:

Flag	Description
-i, --input, --dem	Input raster DEM file
-o, --output	Output raster file
--filter	Filter size (cells)
--slope	Slope threshold value

Python function:

```
remove_off_terrain_objects(
    dem,
    output,
```

```
    filter=11,
    slope=15.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RemoveOffTerrainObjects -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=bare_earth_DEM.tif ^
--filter=25 --slope=10.0
```

7.6.38 RuggednessIndex

Calculates the Riley et al.'s (1999) terrain ruggedness index from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
ruggedness_index(
    dem,
    output,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RuggednessIndex -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.6.39 SedimentTransportIndex

Calculates the sediment transport index.

Parameters:

Flag	Description
--sca	Input raster specific contributing area (SCA) file
--slope	Input raster slope file

Flag	Description
-o, --output	Output raster file
--sca_exponent	SCA exponent value
--slope_exponent	Slope exponent value

Python function:

```
sediment_transport_index(
    sca,
    slope,
    output,
    sca_exponent=0.4,
    slope_exponent=1.3,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SedimentTransportIndex -v ^
--wd="/path/to/data/" --sca='flow_accum.tif' ^
--slope='slope.tif' -o=output.tif --sca_exponent=0.5 ^
--slope_exponent=1.0
```

7.6.40 Slope

Calculates a slope raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
slope(
    dem,
    output,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Slope -v --wd="/path/to/data/" ^
```

```
--dem=DEM.tif -o=output.tif
```

7.6.41 SlopeVsElevationPlot

Creates a slope vs. elevation plot for one or more DEMs.

Parameters:

Flag	Description
-i, --inputs	Input DEM files
--watershed	Input watershed files (optional)
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
slope_vs_elevation_plot(
    inputs,
    output,
    watershed=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SlopeVsElevationPlot -v ^
--wd="/path/to/data/" -i="DEM1.tif;DEM2.tif" ^
--watershed="ws1.tif;ws2.tif" -o=outfile.html
```

7.6.42 TangentialCurvature

Calculates a tangential curvature raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
tangential_curvature(
    dem,
```

```
    output,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TangentialCurvature -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.6.43 TotalCurvature

Calculates a total curvature raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zfactor	Optional multiplier for when the vertical and horizontal units are not the same

Python function:

```
total_curvature(
    dem,
    output,
    zfactor=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TotalCurvature -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.6.44 Viewshed

Identifies the viewshed for a point or set of points.

Parameters:

Flag	Description
--dem	Input raster DEM file
--stations	Input viewing station vector file
-o, --output	Output raster file

Flag	Description
--height	Viewing station height, in z units

Python function:

```
viewshed(
    dem,
    stations,
    output,
    height=2.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Viewshed -v --wd="/path/to/data/" ^
--dem='dem.tif' --stations='stations.shp' -o=output.tif ^
--height=10.0
```

7.6.45 VisibilityIndex

Estimates the relative visibility of sites in a DEM.

Parameters:

Flag	Description
--dem	Input raster DEM file
-o, --output	Output raster file
--height	Viewing station height, in z units
--res_factor	The resolution factor determines the density of measured viewsheds

Python function:

```
visibility_index(
    dem,
    output,
    height=2.0,
    res_factor=2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=VisibilityIndex -v ^
--wd="/path/to/data/" --dem=dem.tif -o=output.tif ^
--height=10.0 --res_factor=4
```

7.6.46 WetnessIndex

Calculates the topographic wetness index, Ln(A / tan(slope)).

Parameters:

Flag	Description
--sca	Input raster specific contributing area (SCA) file
--slope	Input raster slope file
-o, --output	Output raster file

Python function:

```
wetness_index(
    sca,
    slope,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=WetnessIndex -v --wd="/path/to/data/" ^
--sca='flow_accum.tif' --slope='slope.tif' -o=output.tif
```

7.7 Hydrological Analysis

7.7.1 AverageFlowpathSlope

Measures the average slope gradient from each grid cell to all upslope divide cells.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
average_flowpath_slope(
    dem,
    output,
```

```
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AverageFlowpathSlope -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.7.2 AverageUpslopeFlowpathLength

Measures the average length of all upslope flowpaths draining each grid cell.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
average_upslope_flowpath_length(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AverageUpslopeFlowpathLength -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.7.3 Basins

Identifies drainage basins that drain to the DEM edge.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
basins(
    d8_pntr,
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Basins -v --wd="/path/to/data/" ^
--d8_pntr='d8pntr.tif' -o='output.tif'
```

7.7.4 BreachDepressions

Breaches all of the depressions in a DEM using Lindsay's (2016) algorithm. This should be preferred over depression filling in most cases.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--max_depth	Optional maximum breach depth (default is Inf)
--max_length	Optional maximum breach channel length (in grid cells; default is Inf)

Python function:

```
breach_depressions(
    dem,
    output,
    max_depth=None,
    max_length=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=BreachDepressions -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.7.5 BreachSingleCellPits

Removes single-cell pits from an input DEM by breaching.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
breach_single_cell_pits(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=BreachSingleCellPits -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif
```

7.7.6 D8FlowAccumulation

Calculates a D8 flow accumulation raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--out_type	Output type; one of 'cells', 'specific contributing area' (default), and 'catchment area'
--log	Optional flag to request the output be log-transformed
--clip	Optional flag to request clipping the display max by 1%

Python function:

```
d8_flow_accumulation(
    dem,
    output,
    out_type="specific contributing area",
    log=False,
    clip=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=D8FlowAccumulation -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.dtifep ^
```

```
--out_type='cells'
>>./whitebox_tools -r=D8FlowAccumulation -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--out_type='specific catchment area' --log --clip
```

7.7.7 D8MassFlux

Performs a D8 mass flux calculation.

Parameters:

Flag	Description
--dem	Input raster DEM file
--loading	Input loading raster file
--efficiency	Input efficiency raster file
--absorption	Input absorption raster file
-o, --output	Output raster file

Python function:

```
d8_mass_flux(
    dem,
    loading,
    efficiency,
    absorption,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=D8MassFlux -v --wd="/path/to/data/" ^
--dem=DEM.tif --loading=load.tif --efficiency=eff.tif ^
--absorption=abs.tif -o=output.tif
```

7.7.8 D8Pointer

Calculates a D8 flow pointer raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file

Flag	Description
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
d8_pointer(
    dem,
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=D8Pointer -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.7.9 DInfFlowAccumulation

Calculates a D-infinity flow accumulation raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--out_type	Output type; one of 'cells', 'sca' (default), and 'ca'
--threshold	Optional convergence threshold parameter, in grid cells; default is inifinity
--log	Optional flag to request the output be log-transformed
--clip	Optional flag to request clipping the display max by 1%

Python function:

```
d_inf_flow_accumulation(
    dem,
    output,
    out_type="Specific Contributing Area",
    threshold=None,
    log=False,
    clip=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DInfFlowAccumulation -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--out_type=sca
>>./whitebox_tools -r=DInfFlowAccumulation -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--out_type=sca --threshold=10000 --log --clip
```

7.7.10 DInfMassFlux

Performs a D-infinity mass flux calculation.

Parameters:

Flag	Description
--dem	Input raster DEM file
--loading	Input loading raster file
--efficiency	Input efficiency raster file
--absorption	Input absorption raster file
-o, --output	Output raster file

Python function:

```
d_inf_mass_flux(
    dem,
    loading,
    efficiency,
    absorption,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DInfMassFlux -v --wd="/path/to/data/" ^
--dem=DEM.tif --loading=load.tif --efficiency=eff.tif ^
--absorption=abs.tif -o=output.tif
```

7.7.11 DInfPointer

Calculates a D-infinity flow pointer (flow direction) raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
d_inf_pointer(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DInfPointer -v --wd="/path/to/data/" ^
--dem=DEM.tif
```

7.7.12 DepthInSink

Measures the depth of sinks (depressions) in a DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zero_background	Flag indicating whether the background value of zero should be used

Python function:

```
depth_in_sink(
    dem,
    output,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DepthInSink -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif --zero_background
```

7.7.13 DownslopeDistanceToStream

Measures distance to the nearest downslope stream cell.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--streams	Input raster streams file
-o, --output	Output raster file

Python function:

```
downslope_distance_to_stream(
    dem,
    streams,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DownslopeDistanceToStream -v ^
--wd="/path/to/data/" --dem='dem.tif' --streams='streams.tif' ^
-o='output.tif'
```

7.7.14 DownslopeFlowpathLength

Calculates the downslope flowpath length from each cell to basin outlet.

Parameters:

Flag	Description
--d8_pntr	Input D8 pointer raster file
--watersheds	Optional input watershed raster file
--weights	Optional input weights raster file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
downslope_flowpath_length(
    d8_pntr,
    output,
    watersheds=None,
```

```
    weights=None,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DownslopeFlowpathLength -v ^
--wd="/path/to/data/" --d8_pntr=pointer.tif ^
-o=flowpath_len.tif
>>./whitebox_tools ^
-r=DownslopeFlowpathLength -v --wd="/path/to/data/" ^
--d8_pntr=pointer.tif --watersheds=basin.tif ^
--weights=weights.tif -o=flowpath_len.tif --esri_pntr
```

7.7.15 ElevationAboveStream

Calculates the elevation of cells above the nearest downslope stream cell.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--streams	Input raster streams file
-o, --output	Output raster file

Python function:

```
elevation_above_stream(
    dem,
    streams,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ElevationAboveStream -v ^
--wd="/path/to/data/" --dem='dem.tif' --streams='streams.tif' ^
-o='output.tif'
```

7.7.16 ElevationAboveStreamEuclidean

Calculates the elevation of cells above the nearest (Euclidean distance) stream cell.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--streams	Input raster streams file
-o, --output	Output raster file

Python function:

```
elevation_above_stream_euclidean(
    dem,
    streams,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ElevationAboveStreamEuclidean -v ^
--wd="/path/to/data/" -i=DEM.tif --streams=streams.tif ^
-o=output.tif
```

7.7.17 FD8FlowAccumulation

Calculates an FD8 flow accumulation raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--out_type	Output type; one of 'cells', 'specific contributing area' (default), and 'catchment area'
--exponent	Optional exponent parameter; default is 1.1
--threshold	Optional convergence threshold parameter, in grid cells; default is inifinity
--log	Optional flag to request the output be log-transformed
--clip	Optional flag to request clipping the display max by 1%

Python function:

```
fd8_flow_accumulation(
    dem,
    output,
    out_type="specific contributing area",
```

```
    exponent=1.1,
    threshold=None,
    log=False,
    clip=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FD8FlowAccumulation -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--out_type='cells'
>>./whitebox_tools -r=FD8FlowAccumulation -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--out_type='catchment area' --exponent=1.5 --threshold=10000 ^
--log --clip
```

7.7.18 FD8Pointer

Calculates an FD8 flow pointer raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
fd8_pointer(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FD8Pointer -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.7.19 FillBurn

Burns streams into a DEM using the FillBurn (Saunders, 1999) method.

Parameters:

Flag	Description
--dem	Input raster DEM file
--streams	Input vector streams file
-o, --output	Output raster file

Python function:

```
fill_burn(
    dem,
    streams,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FillBurn -v --wd="/path/to/data/" ^
--dem=DEM.tif --streams=streams.shp -o=dem_burned.tif
```

7.7.20 FillDepressions

Fills all of the depressions in a DEM. Depression breaching should be preferred in most cases.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--fix_flats	Optional flag indicating whether flat areas should have a small gradient applied

Python function:

```
fill_depressions(
    dem,
    output,
    fix_flats=True,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FillDepressions -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif ^
--fix_flats
```

7.7.21 FillSingleCellPits

Raises pit cells to the elevation of their lowest neighbour.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
fill_single_cell_pits(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FillSingleCellPits -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=NewRaster.tif
```

7.7.22 FindNoFlowCells

Finds grid cells with no downslope neighbours.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
find_no_flow_cells(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FindNoFlowCells -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=NewRaster.tif
```

7.7.23 FindParallelFlow

Finds areas of parallel flow in D8 flow direction rasters.

Parameters:

Flag	Description
--d8_pntr	Input D8 pointer raster file
--streams	Input raster streams file
-o, --output	Output raster file

Python function:

```
find_parallel_flow(
    d8_pntr,
    streams,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FindParallelFlow -v ^
--wd="/path/to/data/" --d8_pntr=pointer.tif ^
-o=out.tif
>>./whitebox_tools -r=FindParallelFlow -v ^
--wd="/path/to/data/" --d8_pntr=pointer.tif -o=out.tif ^
--streams='streams.tif'
```

7.7.24 FlattenLakes

Flattens lake polygons in a raster DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--lakes	Input lakes vector polygons file
-o, --output	Output raster file

Python function:

```
flatten_lakes(
    dem,
    lakes,
```

```
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FlattenLakes -v --wd="/path/to/data/" ^
--dem='DEM.tif' --lakes='lakes.shp' -o='output.tif'
```

7.7.25 FloodOrder

Assigns each DEM grid cell its order in the sequence of inundations that are encountered during a search starting from the edges, moving inward at increasing elevations.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
flood_order(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FloodOrder -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.7.26 FlowAccumulationFullWorkflow

Resolves all of the depressions in a DEM, outputting a breached DEM, an aspect-aligned non-divergent flow pointer, a flow accumulation raster.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
--out_dem	Output raster DEM file
--out_pntr	Output raster flow pointer file
--out_accum	Output raster flow accumulation file

Flag	Description
--out_type	Output type; one of 'cells', 'sca' (default), and 'ca'
--log	Optional flag to request the output be log-transformed
--clip	Optional flag to request clipping the display max by 1%
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
flow_accumulation_full_workflow(
    dem,
    out_dem,
    out_pntr,
    out_accum,
    out_type="Specific Contributing Area",
    log=False,
    clip=False,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FlowAccumulationFullWorkflow -v ^
--wd="/path/to/data/" --dem='DEM.tif' ^
--out_dem='DEM_filled.tif' --out_pntr='pointer.tif' ^
--out_accum='accum.tif' --out_type=sca --log --clip
```

7.7.27 FlowLengthDiff

Calculates the local maximum absolute difference in downslope flowpath length, useful in mapping drainage divides and ridges.

Parameters:

Flag	Description
--d8_pntr	Input D8 pointer raster file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
flow_length_diff(
    d8_pntr,
    output,
```

```
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FlowLengthDiff -v --wd="/path/to/data/" ^
--d8_pntr=pointer.tif -o=output.tif
```

7.7.28 Hillslopes

Identifies the individual hillslopes draining to each link in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
hillslopes(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Hillslopes -v --wd="/path/to/data/" ^
--d8_pntr='d8pntr.tif' --streams='streams.tif' ^
-o='output.tif'
```

7.7.29 Isobasins

Divides a landscape into nearly equal sized drainage basins (i.e. watersheds).

Parameters:

Flag	Description
-i, --dem	Input raster DEM file

Flag	Description
-o, --output	Output raster file
--size	Target basin size, in grid cells

Python function:

```
isobasins(
    dem,
    output,
    size,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Isobasins -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif --size=1000
```

7.7.30 JensonSnapPourPoints

Moves outlet points used to specify points of interest in a watershedding operation to the nearest stream cell.

Parameters:

Flag	Description
--pour_pts	Input raster pour points (outlet) file
--streams	Input raster streams file
-o, --output	Output raster file
--snap_dist	Maximum snap distance in map units

Python function:

```
jenson_snap_pour_points(
    pour_pts,
    streams,
    output,
    snap_dist,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=JensonSnapPourPoints -v ^
--wd="/path/to/data/" --pour_pts='pour_pts.tif' ^
--streams='streams.tif' -o='output.tif' --snap_dist=15.0
```

7.7.31 MaxUpslopeFlowpathLength

Measures the maximum length of all upslope flowpaths draining each grid cell.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
max_upslope_flowpath_length(
    dem,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaxUpslopeFlowpathLength -v ^
--wd="/path/to/data/" -i=DEM.tif ^
-o=output.tif
>>./whitebox_tools -r=MaxUpslopeFlowpathLength -v ^
--wd="/path/to/data/" --dem=DEM.tif -o=output.tif --log ^
--clip
```

7.7.32 NumInflowingNeighbours

Computes the number of inflowing neighbours to each cell in an input DEM based on the D8 algorithm.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file

Python function:

```
num_inflowing_neighbours(
    dem,
    output,
```

```
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=NumInflowingNeighbours -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.7.33 Rho8Pointer

Calculates a stochastic Rho8 flow pointer raster from an input DEM.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
rho8_pointer(
    dem,
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Rho8Pointer -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif
```

7.7.34 Sink

Identifies the depressions in a DEM, giving each feature a unique identifier.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
sink(
    dem,
    output,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Sink -v --wd="/path/to/data/" ^
--dem=DEM.tif -o=output.tif --zero_background
```

7.7.35 SnapPourPoints

Moves outlet points used to specify points of interest in a watershedding operation to the cell with the highest flow accumulation in its neighbourhood.

Parameters:

Flag	Description
--pour_pts	Input raster pour points (outlet) file
--flow_accum	Input raster D8 flow accumulation file
-o, --output	Output raster file
--snap_dist	Maximum snap distance in map units

Python function:

```
snap_pour_points(
    pour_pts,
    flow_accum,
    output,
    snap_dist,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SnapPourPoints -v --wd="/path/to/data/" ^
--pour_pts='pour_pts.tif' --flow_accum='d8accum.tif' ^
-o='output.tif' --snap_dist=15.0
```

7.7.36 StrahlerOrderBasins

Identifies Strahler-order basins from an input stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
strahler_order_basins(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StrahlerOrderBasins -v ^
--wd="/path/to/data/" --d8_pntr='d8pntr.tif' ^
--streams='streams.tif' -o='output.tif'
```

7.7.37 Subbasins

Identifies the catchments, or sub-basin, draining to each link in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input D8 pointer raster file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
subbasins(
    d8_pntr,
    streams,
```

```
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Subbasins -v --wd="/path/to/data/" ^
--d8_pntr='d8pntr.tif' --streams='streams.tif' ^
-o='output.tif'
```

7.7.38 TraceDownslopeFlowpaths

Traces downslope flowpaths from one or more target sites (i.e. seed points).

Parameters:

Flag	Description
--seed_pts	Input raster seed points file
--d8_pntr	Input D8 pointer raster file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
trace_downslope_flowpaths(
    seed_pts,
    d8_pntr,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TraceDownslopeFlowpaths -v ^
--wd="/path/to/data/" --seed_pts=seeds.tif ^
--flow_dir=flow_directions.tif --output=flow_paths.tif
```

7.7.39 Watershed

Identifies the watershed, or drainage basin, draining to a set of target cells.

Parameters:

Flag	Description
--d8_pntr	Input D8 pointer raster file
--pour_pts	Input vector pour points (outlet) file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
watershed(
    d8_pntr,
    pour_pts,
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Watershed -v --wd="/path/to/data/" ^
--d8_pntr='d8pntr.tif' --pour_pts='pour_pts.shp' ^
-o='output.tif'
```

7.8 Image Processing Tools

7.8.1 Closing

A closing is a mathematical morphology operating involving an erosion (min filter) of a dilation (max filter) set.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
closing(
    i,
    output,
```

```
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Closing -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.8.2 CreateColourComposite

Creates a colour-composite image from three bands of multispectral imagery.

Parameters:

Flag	Description
--red	Input red band image file
--green	Input green band image file
--blue	Input blue band image file
--opacity	Input opacity band image file (optional)
-o, --output	Output colour composite file
--enhance	Optional flag indicating whether a balance contrast enhancement is performed

Python function:

```
create_colour_composite(
    red,
    green,
    blue,
    output,
    opacity=None,
    enhance=True,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CreateColourComposite -v ^
--wd="/path/to/data/" --red=band3.tif --green=band2.tif ^
--blue=band1.tif -o=output.tif
>>./whitebox_tools ^
-r=CreateColourComposite -v --wd="/path/to/data/" ^
--red=band3.tif --green=band2.tif --blue=band1.tif ^
--opacity=a.tif -o=output.tif
```

7.8.3 FlipImage

Reflects an image in the vertical or horizontal axis.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--direction	Direction of reflection; options include 'v' (vertical), 'h' (horizontal), and 'b' (both)

Python function:

```
flip_image(
    i,
    output,
    direction="vertical",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FlipImage -v --wd="/path/to/data/" ^
--input=in.tif -o=out.tif --direction=h
```

7.8.4 IhsToRgb

Converts intensity, hue, and saturation (IHS) images into red, green, and blue (RGB) images.

Parameters:

Flag	Description
--intensity	Input intensity file
--hue	Input hue file
--saturation	Input saturation file
--red	Output red band file. Optionally specified if colour-composite not specified
--green	Output green band file. Optionally specified if colour-composite not specified
--blue	Output blue band file. Optionally specified if colour-composite not specified
-o, --output	Output colour-composite file. Only used if individual bands are not specified

Python function:

```
ihs_to_rgb(
    intensity,
    hue,
    saturation,
    red=None,
    green=None,
    blue=None,
    output=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=IhsToRgb -v --wd="/path/to/data/" ^
--intensity=intensity.tif --hue=hue.tif ^
--saturation=saturation.tif --red=band3.tif --green=band2.tif ^
--blue=band1.tif
>>./whitebox_tools -r=IhsToRgb -v ^
--wd="/path/to/data/" --intensity=intensity.tif --hue=hue.tif ^
--saturation=saturation.tif --composite=image.tif
```

7.8.5 ImageStackProfile

Plots an image stack profile (i.e. signature) for a set of points and multispectral images.

Parameters:

Flag	Description
-i, --inputs	Input multispectral image files
--points	Input vector points file
-o, --output	Output HTML file

Python function:

```
image_stack_profile(
    inputs,
    points,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ImageStackProfile -v ^
--wd="/path/to/data/" -i='image1.tif;image2.tif;image3.tif' ^
--points=pts.shp -o=output.html
```

7.8.6 IntegralImage

Transforms an input image (summed area table) into its integral image equivalent.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
integral_image(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=IntegralImage -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif
```

7.8.7 KMeansClustering

Performs a k-means clustering operation on a multi-spectral dataset.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file
--out_html	Output HTML report file
--classes	Number of classes
--max_iterations	Maximum number of iterations
--class_change	Minimum percent of cells changed between iterations before completion
--initialize	How to initialize cluster centres?
--min_class_size	Minimum class size, in pixels

Python function:

```
k_means_clustering(
    inputs,
    output,
    classes,
    out_html=None,
    max_iterations=10,
    class_change=2.0,
    initialize="diagonal",
    min_class_size=10,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=KMeansClustering -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
-o=output.tif --out_html=report.html --classes=15 ^
--max_iterations=25 --class_change=1.5 --initialize='random' ^
--min_class_size=500
```

7.8.8 LineThinning

Performs line thinning a on Boolean raster image; intended to be used with the RemoveSpurs tool.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
line_thinning(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LineThinning -v --wd="/path/to/data/" ^
--input=DEM.tif -o=output.tif
```

7.8.9 ModifiedKMeansClustering

Performs a modified k-means clustering operation on a multi-spectral dataset.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file
--out_html	Output HTML report file
--start_clusters	Initial number of clusters
--merger_dist	Cluster merger distance
--max_iterations	Maximum number of iterations
--class_change	Minimum percent of cells changed between iterations before completion

Python function:

```
modified_k_means_clustering(
    inputs,
    output,
    out_html=None,
    start_clusters=1000,
    merger_dist=None,
    max_iterations=10,
    class_change=2.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ModifiedKMeansClustering -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
-o=output.tif --out_html=report.html --start_clusters=100 ^
--merger_dist=30.0 --max_iterations=25 --class_change=1.5
```

7.8.10 Mosaic

Mosaics two or more images together.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output raster file
--method	Resampling method

Python function:

```
mosaic(
    inputs,
    output,
    method="cc",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Mosaic -v --wd='/path/to/data/' ^
-i='image1.tif;image2.tif;image3.tif' -o=dest.tif ^
--method='cc
```

7.8.11 NormalizedDifferenceVegetationIndex

Calculates the normalized difference vegetation index (NDVI) from near-infrared and red imagery.

Parameters:

Flag	Description
--nir	Input near-infrared band image
--red	Input red band image
-o, --output	Output raster file
--clip	Optional amount to clip the distribution tails by, in percent
--osavi	Optional flag indicating whether the optimized soil-adjusted veg index (OSAVI) should be used

Python function:

```
normalized_difference_vegetation_index(
    nir,
    red,
    output,
    clip=0.0,
    osavi=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=NormalizedDifferenceVegetationIndex -v ^
--wd="/path/to/data/" --nir=band4.tif --red=band3.tif ^
-o=output.tif
>>./whitebox_tools ^
-r=NormalizedDifferenceVegetationIndex -v --wd="/path/to/data/" ^
```

```
--nir=band4.tif --red=band3.tif -o=output.tif --clip=1.0 ^
--osavi
```

7.8.12 Opening

An opening is a mathematical morphology operating involving a dilation (max filter) of an erosion (min filter) set.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
opening(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Opening -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.8.13 RemoveSpurs

Removes the spurs (pruning operation) from a Boolean line image.; intended to be used on the output of the LineThinning tool.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--iterations	Maximum number of iterations

Python function:

```
remove_spurs(
    i,
    output,
    iterations=10,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RemoveSpurs -v --wd="/path/to/data/" ^
--input=DEM.tif -o=output.tif --iterations=10
```

7.8.14 Resample

Resamples one or more input images into a destination image.

Parameters:

Flag	Description
-i, --inputs	Input raster files
--destination	Destination raster file
--method	Resampling method

Python function:

```
resample(
    inputs,
    destination,
    method="cc",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Resample -v --wd='/path/to/data/' ^
-i='image1.tif;image2.tif;image3.tif' --destination=dest.tif ^
--method='cc
```

7.8.15 RgbToIhs

Converts red, green, and blue (RGB) images into intensity, hue, and saturation (IHS) images.

Parameters:

Flag	Description
--red	Input red band image file. Optionally specified if colour-composite not specified
--green	Input green band image file. Optionally specified if colour-composite not specified
--blue	Input blue band image file. Optionally specified if colour-composite not specified
--composite	Input colour-composite image file. Only used if individual bands are not specified
--intensity	Output intensity raster file
--hue	Output hue raster file
--saturation	Output saturation raster file

Python function:

```
rgb_to_ihs(
    intensity,
    hue,
    saturation,
    red=None,
    green=None,
    blue=None,
    composite=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RgbToIhs -v --wd="/path/to/data/" ^
--red=band3.tif --green=band2.tif --blue=band1.tif ^
--intensity=intensity.tif --hue=hue.tif ^
--saturation=saturation.tif
>>./whitebox_tools -r=RgbToIhs -v ^
--wd="/path/to/data/" --composite=image.tif ^
--intensity=intensity.tif --hue=hue.tif ^
--saturation=saturation.tif
```

7.8.16 SplitColourComposite

This tool splits an RGB colour composite image into seperate multispectral images.

Parameters:

Flag	Description
-i, --input	Input colour composite image file

Flag	Description
-o, --output	Output raster file (suffixes of '_r', '_g', and '_b' will be appended)

Python function:

```
split_colour_composite(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SplitColourComposite -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif
```

7.8.17 ThickenRasterLine

Thickens single-cell wide lines within a raster image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
thicken_raster_line(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ThickenRasterLine -v ^
--wd="/path/to/data/" --input=DEM.tif -o=output.tif
```

7.8.18 TophatTransform

Performs either a white or black top-hat transform on an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction
--variant	Optional variant value. Options include 'white' and 'black'

Python function:

```
tophat_transform(
    i,
    output,
    filterx=11,
    filtery=11,
    variant="white",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TophatTransform -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --filter=25
```

7.8.19 WriteFunctionMemoryInsertion

Performs a write function memory insertion for single-band multi-date change detection.

Parameters:

Flag	Description
--i1, --input1	Input raster file associated with the first date
--i2, --input2	Input raster file associated with the second date
--i3, --input3	Optional input raster file associated with the third date
-o, --output	Output raster file

Python function:

```
write_function_memory_insertion(
    input1,
    input2,
    output,
    input3=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=WriteFunctionMemoryInsertion -v ^
--wd="/path/to/data/" -i1=input1.tif -i2=input2.tif ^
-o=output.tif
```

7.9 Image Processing Tools => Filters

7.9.1 AdaptiveFilter

Performs an adaptive filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction
--threshold	Difference from mean threshold, in standard deviations

Python function:

```
adaptive_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    threshold=2.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AdaptiveFilter -v --wd="/path/to/data/" ^
-i=DEM.tif -o=output.tif --filter=25 --threshold = 2.0
```

7.9.2 BilateralFilter

A bilateral filter is an edge-preserving smoothing filter introduced by Tomasi and Manduchi (1998).

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--sigma_dist	Standard deviation in distance in pixels
--sigma_int	Standard deviation in intensity in pixels

Python function:

```
bilateral_filter(
    i,
    output,
    sigma_dist=0.75,
    sigma_int=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=BilateralFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif ^
--sigma_dist=2.5 --sigma_int=4.0
```

7.9.3 ConservativeSmoothingFilter

Performs a conservative-smoothing filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
conservative_smoothing_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ConservativeSmoothingFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --filter=25
```

7.9.4 DiffOfGaussianFilter

Performs a Difference of Gaussian (DoG) filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--sigma1	Standard deviation distance in pixels
--sigma2	Standard deviation distance in pixels

Python function:

```
diff_of_gaussian_filter(
    i,
    output,
    sigma1=2.0,
    sigma2=4.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DiffOfGaussianFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --sigma1=2.0 ^
--sigma2=4.0
```

7.9.5 DiversityFilter

Assigns each cell in the output grid the number of different values in a moving window centred on each grid cell in the input raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
diversity_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DiversityFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --filter=25
```

7.9.6 EdgePreservingMeanFilter

Performs a simple edge-preserving mean filter on an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filter	Size of the filter kernel
--threshold	Maximum difference in values

Python function:

```
edge_preserving_mean_filter(
    i,
    output,
    threshold,
    filter=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=EdgePreservingMeanFilter -v ^
--wd="/path/to/data/" --input=image.tif -o=output.tif ^
--filter=5 --threshold=20
```

7.9.7 EmbossFilter

Performs an emboss filter on an image, similar to a hillshade operation.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--direction	Direction of reflection; options include 'n', 's', 'e', 'w', 'ne', 'se', 'nw', 'sw'
--clip	Optional amount to clip the distribution tails by, in percent

Python function:

```
emboss_filter(
    i,
    output,
    direction="n",
    clip=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=EmbossFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --direction='s' --clip=1.0
```

7.9.8 GaussianFilter

Performs a Gaussian filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--sigma	Standard deviation distance in pixels

Python function:

```
gaussian_filter(
    i,
    output,
    sigma=0.75,
```

```
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=GaussianFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --sigma=2.0
```

7.9.9 HighPassFilter

Performs a high-pass filter on an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
high_pass_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HighPassFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.9.10 KNearestMeanFilter

A k-nearest mean filter is a type of edge-preserving smoothing filter.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction

Flag	Description
--filtery	Size of the filter kernel in the y-direction
-k	k-value in pixels; this is the number of nearest-valued neighbours to use

Python function:

```
k_nearest_mean_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    k=5,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=KNearestMeanFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --filter=9 ^
-k=5
>>./whitebox_tools -r=KNearestMeanFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --filtery=7 ^
--filtery=9 -k=5
```

7.9.11 LaplacianFilter

Performs a Laplacian filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--variant	Optional variant value. Options include 3x3(1), 3x3(2), 3x3(3), 3x3(4), 5x5(1), and 5x5(2) (default is 3x3(1))
--clip	Optional amount to clip the distribution tails by, in percent

Python function:

```
laplacian_filter(
    i,
    output,
    variant="3x3(1)",
    clip=0.0,
```

```
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LaplacianFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif ^
--variant='3x3(1)' --clip=1.0
```

7.9.12 LaplacianOfGaussianFilter

Performs a Laplacian-of-Gaussian (LoG) filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--sigma	Standard deviation in pixels

Python function:

```
laplacian_of_gaussian_filter(
    i,
    output,
    sigma=0.75,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LaplacianOfGaussianFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --sigma=2.0
```

7.9.13 LeeFilter

Performs a Lee (Sigma) smoothing filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Flag	Description
--sigma	Sigma value should be related to the standarad deviation of the distribution of image speckle noise
-m	M-threshold value the minimum allowable number of pixels within the intensity range

Python function:

```
lee_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    sigma=10.0,
    m=5.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LeeFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=9 --sigma=10.0 ^
-m=5
>>./whitebox_tools -r=LeeFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filtery=7 --filtery=9 ^
--sigma=10.0 -m=5
```

7.9.14 LineDetectionFilter

Performs a line-detection filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--variant	Optional variant value. Options include 'v' (vertical), 'h' (horizontal), '45', and '135' (default is 'v')
--absvals	Optional flag indicating whether outputs should be absolute values
--clip	Optional amount to clip the distribution tails by, in percent

Python function:

```
line_detection_filter(
    i,
    output,
    variant="vertical",
    absvals=False,
    clip=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LineDetectionFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --variant=h ^
--clip=1.0
```

7.9.15 MajorityFilter

Assigns each cell in the output grid the most frequently occurring value (mode) in a moving window centred on each grid cell in the input raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
majority_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MajorityFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.9.16 MaximumFilter

Assigns each cell in the output grid the maximum value in a moving window centred on each grid cell in the input raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
maximum_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MaximumFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.9.17 MeanFilter

Performs a mean filter (low-pass filter) on an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
mean_filter(
    i,
    output,
```

```
    filterx=3,
    filtery=3,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MeanFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filterx=25 --filtery=25
```

7.9.18 MedianFilter

Performs a median filter on an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction
--sig_digits	Number of significant digits

Python function:

```
median_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    sig_digits=2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MedianFilter -v --wd="/path/to/data/" ^
-i=input.tif -o=output.tif --filter=25
```

7.9.19 MinimumFilter

Assigns each cell in the output grid the minimum value in a moving window centred on each grid cell in the input raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
minimum_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MinimumFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.9.20 OlympicFilter

Performs an olympic smoothing filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
olympic_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=OlympicFilter -v --wd="/path/to/data/" ^
```

```
-i=image.tif -o=output.tif --filter=25
```

7.9.21 PercentileFilter

Performs a percentile filter on an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction
--sig_digits	Number of significant digits

Python function:

```
percentile_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    sig_digits=2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PercentileFilter -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif --filter=25
```

7.9.22 PrewittFilter

Performs a Prewitt edge-detection filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--clip	Optional amount to clip the distribution tails by, in percent

Python function:

```
prewitt_filter(
    i,
    output,
    clip=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PrewittFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --clip=1.0
```

7.9.23 RangeFilter

Assigns each cell in the output grid the range of values in a moving window centred on each grid cell in the input raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
range_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RangeFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.9.24 RobertsCrossFilter

Performs a Robert's cross edge-detection filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--clip	Optional amount to clip the distribution tails by, in percent

Python function:

```
roberts_cross_filter(
    i,
    output,
    clip=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RobertsCrossFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --clip=1.0
```

7.9.25 ScharrFilter

Performs a Scharr edge-detection filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--clip	Optional amount to clip the distribution tails by, in percent

Python function:

```
scharr_filter(
    i,
    output,
    clip=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ScharrFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --clip=1.0
```

7.9.26 SobelFilter

Performs a Sobel edge-detection filter on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--variant	Optional variant value. Options include 3x3 and 5x5 (default is 3x3)
--clip	Optional amount to clip the distribution tails by, in percent (default is 0.0)

Python function:

```
sobel_filter(
    i,
    output,
    variant="3x3",
    clip=0.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SobelFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --variant=5x5 --clip=1.0
```

7.9.27 StandardDeviationFilter

Assigns each cell in the output grid the standard deviation of values in a moving window centred on each grid cell in the input raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
standard_deviation_filter(
    i,
    output,
```

```
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StandardDeviationFilter -v ^
--wd="/path/to/data/" -i=image.tif -o=output.tif --filter=25
```

7.9.28 TotalFilter

Performs a total filter on an input image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--filterx	Size of the filter kernel in the x-direction
--filtery	Size of the filter kernel in the y-direction

Python function:

```
total_filter(
    i,
    output,
    filterx=11,
    filtery=11,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TotalFilter -v --wd="/path/to/data/" ^
-i=image.tif -o=output.tif --filter=25
```

7.10 Image Processing Tools => Image Enhancement

7.10.1 BalanceContrastEnhancement

Performs a balance contrast enhancement on a colour-composite image of multispectral data.

Parameters:

Flag	Description
-i, --input	Input colour composite image file
-o, --output	Output raster file
--band_mean	Band mean value

Python function:

```
balance_contrast_enhancement(
    i,
    output,
    band_mean=100.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=BalanceContrastEnhancement -v ^
--wd="/path/to/data/" --input=image.tif -o=output.tif ^
--band_mean=120
```

7.10.2 DirectDecorrelationStretch

Performs a direct decorrelation stretch enhancement on a colour-composite image of multispectral data.

Parameters:

Flag	Description
-i, --input	Input colour composite image file
-o, --output	Output raster file
-k	Achromatic factor (k) ranges between 0 (no effect) and 1 (full saturation stretch), although typical values range from 0.3 to 0.7
--clip	Optional percent to clip the upper tail by during the stretch

Python function:

```
direct_decorrelation_stretch(
    i,
    output,
    k=0.5,
    clip=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DirectDecorrelationStretch -v ^
```

```
--wd="/path/to/data/" --input=image.tif -o=output.tif -k=0.4
```

7.10.3 GammaCorrection

Performs a sigmoidal contrast stretch on input images.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--gamma	Gamma value

Python function:

```
gamma_correction(
    i,
    output,
    gamma=0.5,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=GammaCorrection -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif --gamma=0.5
```

7.10.4 HistogramEqualization

Performs a histogram equalization contrast enhancment on an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--num_tones	Number of tones in the output image

Python function:

```
histogram_equalization(
    i,
    output,
```

```
    num_tones=256,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HistogramEqualization -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif ^
--num_tones=1024
```

7.10.5 HistogramMatching

Alters the statistical distribution of a raster image matching it to a specified PDF.

Parameters:

Flag	Description
-i, --input	Input raster file
--histo_file	Input reference probability distribution function (pdf) text file
-o, --output	Output raster file

Python function:

```
histogram_matching(
    i,
    histo_file,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HistogramMatching -v ^
--wd="/path/to/data/" -i=input1.tif --histo_file=histo.txt ^
-o=output.tif
```

7.10.6 HistogramMatchingTwoImages

This tool alters the cumulative distribution function of a raster image to that of another image.

Parameters:

Flag	Description
--i1, --input1	Input raster file to modify
--i2, --input2	Input reference raster file

Flag	Description
-o, --output	Output raster file

Python function:

```
histogram_matching_two_images(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HistogramMatchingTwoImages -v ^
--wd="/path/to/data/" --i1=input1.tif --i2=input2.tif ^
-o=output.tif
```

7.10.7 MinMaxContrastStretch

Performs a min-max contrast stretch on an input greytone image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--min_val	Lower tail clip value
--max_val	Upper tail clip value
--num_tones	Number of tones in the output image

Python function:

```
min_max_contrast_stretch(
    i,
    output,
    min_val,
    max_val,
    num_tones=256,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=MinMaxContrastStretch -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif ^
```

```
--min_val=45.0 --max_val=200.0 --num_tones=1024
```

7.10.8 PanchromaticSharpening

Increases the spatial resolution of image data by combining multispectral bands with panchromatic data.

Parameters:

Flag	Description
--red	Input red band image file. Optionally specified if colour-composite not specified
--green	Input green band image file. Optionally specified if colour-composite not specified
--blue	Input blue band image file. Optionally specified if colour-composite not specified
--composite	Input colour-composite image file. Only used if individual bands are not specified
--pan	Input panchromatic band file
-o, --output	Output colour composite file
--method	Options include 'brovey' (default) and 'ihs'

Python function:

```
panchromatic_sharpening(
    pan,
    output,
    red=None,
    green=None,
    blue=None,
    composite=None,
    method="brovey",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PanchromaticSharpening -v ^
--wd="/path/to/data/" --red=red.tif --green=green.tif ^
--blue=blue.tif --pan=pan.tif --output=pan_sharp.tif ^
--method='brovey'
>>./whitebox_tools -r=PanchromaticSharpening ^
-v --wd="/path/to/data/" --composite=image.tif --pan=pan.tif ^
--output=pan_sharp.tif --method='ihs'
```

7.10.9 PercentageContrastStretch

Performs a percentage linear contrast stretch on input images.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--clip	Optional amount to clip the distribution tails by, in percent
--tail	Specified which tails to clip; options include 'upper', 'lower', and 'both' (default is 'both')
--num_tones	Number of tones in the output image

Python function:

```
percentage_contrast_stretch(
    i,
    output,
    clip=0.0,
    tail="both",
    num_tones=256,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PercentageContrastStretch -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif --clip=2.0 ^
--tail='both' --num_tones=1024
```

7.10.10 SigmoidalContrastStretch

Performs a sigmoidal contrast stretch on input images.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--cutoff	Cutoff value between 0.0 and 0.95
--gain	Gain value
--num_tones	Number of tones in the output image

Python function:

```
sigmoidal_contrast_stretch(
    i,
    output,
    cutoff=0.0,
    gain=1.0,
    num_tones=256,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SigmoidalContrastStretch -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif --cutoff=0.1 ^
--gain=2.0 --num_tones=1024
```

7.10.11 StandardDeviationContrastStretch

Performs a standard-deviation contrast stretch on input images.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--clip, --stdev	Standard deviation clip value
--num_tones	Number of tones in the output image

Python function:

```
standard_deviation_contrast_stretch(
    i,
    output,
    stdev=2.0,
    num_tones=256,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StandardDeviationContrastStretch -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif --stdev=2.0 ^
--num_tones=1024
```

7.11 LiDAR Tools

7.11.1 BlockMaximum

Creates a block-maximum raster from an input LAS file. When the input/output parameters are not specified, the tool grids all LAS files contained within the working directory.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output file
--resolution	Output raster's grid resolution

Python function:

```
block_maximum(
    i=None,
    output=None,
    resolution=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=BlockMaximum -v --wd="/path/to/data/" ^
-i=file.las -o=outfile.tif --resolution=2.0"
./whitebox_tools ^
-r=BlockMaximum -v --wd="/path/to/data/" -i=file.las ^
-o=outfile.tif --resolution=5.0 --palette=light_quant.plt
```

7.11.2 BlockMinimum

Creates a block-minimum raster from an input LAS file. When the input/output parameters are not specified, the tool grids all LAS files contained within the working directory.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output file
--resolution	Output raster's grid resolution

Python function:

```
block_minimum(
    i=None,
    output=None,
    resolution=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=BlockMinimum -v --wd="/path/to/data/" ^
-i=file.las -o=outfile.tif --resolution=2.0"
./whitebox_tools ^
-r=BlockMinimum -v --wd="/path/to/data/" -i=file.las ^
-o=outfile.tif --resolution=5.0 --palette=light_quant.plt
```

7.11.3 FilterLidarScanAngles

Removes points in a LAS file with scan angles greater than a threshold.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output LiDAR file
--threshold	Scan angle threshold

Python function:

```
filter_lidar_scan_angles(
    i,
    output,
    threshold,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FilterLidarScanAngles -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las" ^
--threshold=10.0
```

7.11.4 FindFlightlineEdgePoints

Identifies points along a flightline's edge in a LAS file.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output file

Python function:

```
find_flightline_edge_points(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FindFlightlineEdgePoints -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las"
```

7.11.5 FlightlineOverlap

Reads a LiDAR (LAS) point file and outputs a raster containing the number of overlapping flight lines in each grid cell.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output file
--resolution	Output raster's grid resolution

Python function:

```
flightline_overlap(
    i=None,
    output=None,
    resolution=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FlightlineOverlap -v ^
--wd="/path/to/data/" -i=file.las -o=outfile.tif ^
--resolution=2.0"
./whitebox_tools -r=FlightlineOverlap -v ^
```

```
--wd="/path/to/data/" -i=file.las -o=outfile.tif ^
--resolution=5.0 --palette=light_quant.plt
```

7.11.6 LasToAscii

Converts one or more LAS files into ASCII text files.

Parameters:

Flag	Description
-i, --inputs	Input LiDAR files

Python function:

```
las_to_ascii(
    inputs,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LasToAscii -v --wd="/path/to/data/" ^
-i="file1.las, file2.las, file3.las" -o=outfile.las"
```

7.11.7 LidarColourize

Adds the red-green-blue colour fields of a LiDAR (LAS) file based on an input image.

Parameters:

Flag	Description
--in_lidar	Input LiDAR file
--in_image	Input colour image file
-o, --output	Output LiDAR file

Python function:

```
lidar_colourize(
    in_lidar,
    in_image,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarColourize -v --wd="/path/to/data/" ^
--in_lidar="input.las" --in_image="image.tif" ^
-o="output.las"
```

7.11.8 LidarElevationSlice

Outputs all of the points within a LiDAR (LAS) point file that lie between a specified elevation range.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output LiDAR file
--minz	Minimum elevation value (optional)
--maxz	Maximum elevation value (optional)
--class	Optional boolean flag indicating whether points outside the range should be retained in output but reclassified
--inclassval	Optional parameter specifying the class value assigned to points within the slice
--outclassval	Optional parameter specifying the class value assigned to points within the slice

Python function:

```
lidar_elevation_slice(
    i,
    output,
    minz=None,
    maxz=None,
    cls=False,
    inclassval=2,
    outclassval=1,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarElevationSlice -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las" ^
--minz=100.0 --maxz=250.0
>>./whitebox_tools ^
-r=LidarElevationSlice -v -i="/path/to/data/input.las" ^
-o="/path/to/data/output.las" --minz=100.0 --maxz=250.0 ^
```

```
--class
>>./whitebox_tools -r=LidarElevationSlice -v ^
-i="/path/to/data/input.las" -o="/path/to/data/output.las" ^
--minz=100.0 --maxz=250.0 --inclassval=1 --outclassval=0
```

7.11.9 LidarGroundPointFilter

Identifies ground points within LiDAR dataset using a slope-based method.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output LiDAR file
--radius	Search Radius
--slope_threshold	Maximum inter-point slope to be considered an off-terrain point
--height_threshold	Inter-point height difference to be considered an off-terrain point

Python function:

```
lidar_ground_point_filter(
    i,
    output,
    radius=2.0,
    slope_threshold=45.0,
    height_threshold=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarGroundPointFilter -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las" ^
--radius=10.0
```

7.11.10 LidarHillshade

Calculates a hillshade value for points within a LAS file and stores these data in the RGB field.

Parameters:

Flag	Description
-i, --input	Input LiDAR file

Flag	Description
-o, --output	Output file
--azimuth	Illumination source azimuth in degrees
--altitude	Illumination source altitude in degrees
--radius	Search Radius

Python function:

```
lidar_hillshade(
    i,
    output,
    azimuth=315.0,
    altitude=30.0,
    radius=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarHillshade -v --wd="/path/to/data/" ^
-i="input.las" -o="output.las" --radius=10.0
>>./whitebox_tools ^
-r=LidarHillshade -v --wd="/path/to/data/" -i="input.las" ^
-o="output.las" --azimuth=180.0 --altitude=20.0 --radius=1.0
```

7.11.11 LidarHistogram

Creates a histogram from LiDAR data.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output HTML file (default name will be based on input file if unspecified)
--parameter	Parameter; options are 'elevation' (default), 'intensity', 'scan angle', 'class
--clip	Amount to clip distribution tails (in percent)

Python function:

```
lidar_histogram(
    i,
    output,
    parameter="elevation",
    clip=1.0,
```

```
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarHistogram -v --wd="/path/to/data/" ^
-i="file1.tif, file2.tif, file3.tif" -o=outfile.htm ^
--contiguity=Bishopsl
```

7.11.12 LidarIdwInterpolation

Interpolates LAS files using an inverse-distance weighted (IDW) scheme. When the input/output parameters are not specified, the tool interpolates all LAS files contained within the working directory.

Parameters:

Flag	Description
-i, --input	Input LiDAR file (including extension)
-o, --output	Output raster file (including extension)
--parameter	Interpolation parameter; options are 'elevation' (default), 'intensity', 'class', 'scan angle', 'user data'
--returns	Point return types to include; options are 'all' (default), 'last', 'first'
--resolution	Output raster's grid resolution
--weight	IDW weight value
--radius	Search Radius
--exclude_cls	Optional exclude classes from interpolation; Valid class values range from 0 to 18, based on LAS specifications. Example, –exclude_cls='3,4,5,6,7,18'
--minz	Optional minimum elevation for inclusion in interpolation
--maxz	Optional maximum elevation for inclusion in interpolation

Python function:

```
lidar_idw_interpolation(
    i=None,
    output=None,
    parameter="elevation",
    returns="all",
    resolution=1.0,
    weight=1.0,
    radius=2.5,
    exclude_cls=None,
    minz=None,
    maxz=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarIdwInterpolation -v ^
--wd="/path/to/data/" -i=file.las -o=outfile.tif ^
--resolution=2.0 --radius=5.0"
./whitebox_tools ^
-r=LidarIdwInterpolation --wd="/path/to/data/" -i=file.las ^
-o=outfile.tif --resolution=5.0 --weight=2.0 --radius=2.0 ^
--exclude_cls='3,4,5,6,7,18' --palette=light_quant.plt
```

7.11.13 LidarInfo

Prints information about a LiDAR (LAS) dataset, including header, point return frequency, and classification data and information about the variable length records (VLRs) and geokeys.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output HTML file for summary report
--vlr	Flag indicating whether or not to print the variable length records (VLRs)
--geokeys	Flag indicating whether or not to print the geokeys

Python function:

```
lidar_info(
    i,
    output=None,
    vlr=False,
    geokeys=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarInfo -v --wd="/path/to/data/" ^
-i=file.las --vlr --geokeys"
./whitebox_tools -r=LidarInfo ^
--wd="/path/to/data/" -i=file.las
```

7.11.14 LidarJoin

Joins multiple LiDAR (LAS) files into a single LAS file.

Parameters:

Flag	Description
-i, --inputs	Input LiDAR files
-o, --output	Output LiDAR file

Python function:

```
lidar_join(
    inputs,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarJoin -v --wd="/path/to/data/" ^
-i="file1.las, file2.las, file3.las" -o=outfile.las"
```

7.11.15 LidarKappaIndex

Performs a kappa index of agreement (KIA) analysis on the classifications of two LAS files.

Parameters:

Flag	Description
--i1, --input1	Input LiDAR classification file
--i2, --input2	Input LiDAR reference file
-o, --output	Output HTML file

Python function:

```
lidar_kappa_index(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarKappaIndex -v ^
--wd="/path/to/data/" --i1=class.tif --i2=reference.tif ^
-o=kia.html
```

7.11.16 LidarNearestNeighbourGridding

Grids LAS files using nearest-neighbour scheme. When the input/output parameters are not specified, the tool grids all LAS files contained within the working directory.

Parameters:

Flag	Description
-i, --input	Input LiDAR file (including extension)
-o, --output	Output raster file (including extension)
--parameter	Interpolation parameter; options are 'elevation' (default), 'intensity', 'class', 'scan angle', 'user data'
--returns	Point return types to include; options are 'all' (default), 'last', 'first'
--resolution	Output raster's grid resolution
--radius	Search Radius
--exclude_cls	Optional exclude classes from interpolation; Valid class values range from 0 to 18, based on LAS specifications. Example, –exclude_cls='3,4,5,6,7,18'
--minz	Optional minimum elevation for inclusion in interpolation
--maxz	Optional maximum elevation for inclusion in interpolation

Python function:

```
lidar_nearest_neighbour_gridding(
    i=None,
    output=None,
    parameter="elevation",
    returns="all",
    resolution=1.0,
    radius=2.5,
    exclude_cls=None,
    minz=None,
    maxz=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarNearestNeighbourGridding -v ^
--wd="/path/to/data/" -i=file.las -o=outfile.tif ^
--resolution=2.0 --radius=5.0"
./whitebox_tools ^
-r=LidarNearestNeighbourGridding --wd="/path/to/data/" ^
-i=file.las -o=outfile.tif --resolution=5.0 --radius=2.0 ^
--exclude_cls='3,4,5,6,7,18' --palette=light_quant.plt
```

7.11.17 LidarPointDensity

Calculates the spatial pattern of point density for a LiDAR data set. When the input/output parameters are not specified, the tool grids all LAS files contained within the working directory.

Parameters:

Flag	Description
-i, --input	Input LiDAR file (including extension)
-o, --output	Output raster file (including extension)
--returns	Point return types to include; options are 'all' (default), 'last', 'first'
--resolution	Output raster's grid resolution
--radius	Search Radius
--exclude_cls	Optional exclude classes from interpolation; Valid class values range from 0 to 18, based on LAS specifications. Example, –exclude_cls='3,4,5,6,7,18'
--minz	Optional minimum elevation for inclusion in interpolation
--maxz	Optional maximum elevation for inclusion in interpolation

Python function:

```
lidar_point_density(
    i=None,
    output=None,
    returns="all",
    resolution=1.0,
    radius=2.5,
    exclude_cls=None,
    minz=None,
    maxz=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarPointDensity -v ^
--wd="/path/to/data/" -i=file.las -o=outfile.tif ^
--resolution=2.0 --radius=5.0"
./whitebox_tools ^
-r=LidarPointDensity -v --wd="/path/to/data/" -i=file.las ^
-o=outfile.tif --resolution=5.0 --radius=2.0 ^
--exclude_cls='3,4,5,6,7,18' --palette=light_quant.plt
```

7.11.18 LidarPointStats

Creates several rasters summarizing the distribution of LAS point data. When the input/output parameters are not specified, the tool works on all LAS files contained within the working directory.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
--resolution	Output raster's grid resolution
--num_points	Flag indicating whether or not to output the number of points raster
--num_pulses	Flag indicating whether or not to output the number of pulses raster
--z_range	Flag indicating whether or not to output the elevation range raster
--intensity_range	Flag indicating whether or not to output the intensity range raster
--predom_class	Flag indicating whether or not to output the predominant classification raster

Python function:

```
lidar_point_stats(
    i=None,
    resolution=1.0,
    num_points=False,
    num_pulses=False,
    z_range=False,
    intensity_range=False,
    predom_class=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarPointStats -v ^
--wd="/path/to/data/" -i=file.las --resolution=1.0 ^
--num_points
```

7.11.19 LidarRemoveDuplicates

Removes duplicate points from a LiDAR data set.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output LiDAR file
--include_z	Include z-values in point comparison?

Python function:

```
lidar_remove_duplicates(
    i,
    output,
    include_z=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarRemoveDuplicates -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las"
```

7.11.20 LidarRemoveOutliers

Removes outliers (high and low points) in a LiDAR point cloud.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output LiDAR file
--radius	Search Radius
--elev_diff	Max. elevation difference

Python function:

```
lidar_remove_outliers(
    i,
    output,
    radius=2.0,
    elev_diff=50.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarRemoveOutliers -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las" ^
--radius=10.0 --elev_diff=25.0
```

7.11.21 LidarSegmentation

Segments a LiDAR point cloud based on normal vectors.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output file
--dist, --radius	Search Radius
--norm_diff	Maximum difference in normal vectors, in degrees
--maxzdiff	Maximum difference in elevation (z units) between neighbouring points of the same segment

Python function:

```
lidar_segmentation(
    i,
    output,
    radius=5.0,
    norm_diff=10.0,
    maxzdiff=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarSegmentation -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las" ^
--radius=10.0 --norm_diff=2.5 --maxzdiff=0.75
```

7.11.22 LidarSegmentationBasedFilter

Identifies ground points within LiDAR point clouds using a segmentation based approach.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output file
--dist, --radius	Search Radius
--norm_diff	Maximum difference in normal vectors, in degrees
--maxzdiff	Maximum difference in elevation (z units) between neighbouring points of the same segment
--classify	Classify points as ground (2) or off-ground (1)

Python function:

```
lidar_segmentation_based_filter(
    i,
    output,
    radius=5.0,
    norm_diff=2.0,
    maxzdiff=1.0,
    classify=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarSegmentationBasedFilter -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las" ^
--radius=10.0 --norm_diff=2.5 --maxzdiff=0.75 --classify
```

7.11.23 LidarTile

Tiles a LiDAR LAS file into multiple LAS files.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
--width_x	Width of tiles in the X dimension; default 1000.0
--width_y	Width of tiles in the Y dimension
--origin_x	Origin point X coordinate for tile grid
--origin_y	Origin point Y coordinate for tile grid
--min_points	Minimum number of points contained in a tile for it to be saved

Python function:

```
lidar_tile(
    i,
    width_x=1000.0,
    width_y=1000.0,
    origin_x=0.0,
    origin_y=0.0,
    min_points=0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarTile -v -i=/path/to/data/input.las ^
--width_x=1000.0 --width_y=2500.0 -=min_points=100
```

7.11.24 LidarTophatTransform

Performs a white top-hat transform on a Lidar dataset; as an estimate of height above ground, this is useful for modelling the vegetation canopy.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output LiDAR file
--radius	Search Radius

Python function:

```
lidar_tophat_transform(
    i,
    output,
    radius=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LidarTophatTransform -v ^
--wd="/path/to/data/" -i="input.las" -o="output.las" ^
--radius=10.0
```

7.11.25 NormalVectors

Calculates normal vectors for points within a LAS file and stores these data (XYZ vector components) in the RGB field.

Parameters:

Flag	Description
-i, --input	Input LiDAR file
-o, --output	Output LiDAR file
--radius	Search Radius

Python function:

```
normal_vectors(
    i,
    output,
    radius=1.0,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=NormalVectors -v --wd="/path/to/data/" ^
-i="input.las" -o="output.las" --radius=10.0
```

7.12 Math and Stats Tools

7.12.1 AbsoluteValue

Calculates the absolute value of every cell in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
absolute_value(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AbsoluteValue -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.2 Add

Performs an addition operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value

Flag	Description
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
add(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Add -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.3 And

Performs a logical AND operator on two Boolean raster images.

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file
-o, --output	Output raster file

Python function:

```
And(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=And -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.4 Anova

Performs an analysis of variance (ANOVA) test on a raster dataset.

Parameters:

Flag	Description
-i, --input	Input raster file
--features	Feature definition (or class) raster
-o, --output	Output HTML file

Python function:

```
anova(
    i,
    features,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Anova -v --wd="/path/to/data/" ^
-i=data.tif --features=classes.tif -o=anova.html
```

7.12.5 ArcCos

Returns the inverse cosine (arccos) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
arc_cos(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ArcCos -v --wd="/path/to/data/" ^
```

```
-i='input.tif' -o=output.tif
```

7.12.6 ArcSin

Returns the inverse sine (arcsin) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
arc_sin(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ArcSin -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.7 ArcTan

Returns the inverse tangent (arctan) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
arc_tan(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ArcTan -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.8 Atan2

Returns the 2-argument inverse tangent (atan2).

Parameters:

Flag	Description
--input_y	Input y raster file or constant value (rise)
--input_x	Input x raster file or constant value (run)
-o, --output	Output raster file

Python function:

```
atan2(
    input_y,
    input_x,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Atan2 -v --wd="/path/to/data/" ^
--input_y='in1.tif' --input_x='in2.tif' -o=output.tif
```

7.12.9 AttributeHistogram

Creates a histogram for the field values of a vector's attribute table.

Parameters:

Flag	Description
-i, --input	Input raster file
--field	Input field name in attribute table
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
attribute_histogram(
    i,
```

```
    field,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AttributeHistogram -v ^
--wd="/path/to/data/" -i=lakes.shp --field=HEIGHT ^
-o=outfile.html
```

7.12.10 AttributeScattergram

Creates a scattergram for two field values of a vector's attribute table.

Parameters:

Flag	Description
-i, --input	Input raster file
--fieldx	Input field name in attribute table for the x-axis
--fieldy	Input field name in attribute table for the y-axis
-o, --output	Output HTML file (default name will be based on input file if unspecified)
--trendline	Draw the trendline

Python function:

```
attribute_scattergram(
    i,
    fieldx,
    fieldy,
    output,
    trendline=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=AttributeScattergram -v ^
--wd="/path/to/data/" -i=lakes.shp --fieldx=HEIGHT ^
--fieldy=area -o=outfile.html --trendline
```

7.12.11 Ceil

Returns the smallest (closest to negative infinity) value that is greater than or equal to the values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
ceil(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Ceil -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.12 Cos

Returns the cosine (cos) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
cos(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Cos -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.13 Cosh

Returns the hyperbolic cosine (cosh) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
cosh(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Cosh -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.14 CrispnessIndex

Calculates the Crispness Index, which is used to quantify how crisp (or conversely how fuzzy) a probability image is.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Optional output html file (default name will be based on input file if unspecified)

Python function:

```
crispness_index(
    i,
    output=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CrispnessIndex -v --wd="/path/to/data/" ^
```

```
-i=input.tif
>>./whitebox_tools -r=CrispnessIndex -v ^
--wd="/path/to/data/" -o=crispness.html
```

7.12.15 CrossTabulation

Performs a cross-tabulation on two categorical images.

Parameters:

Flag	Description
--i1, --input1	Input raster file 1
--i2, --input2	Input raster file 1
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
cross_tabulation(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CrossTabulation -v ^
--wd="/path/to/data/" --i1="file1.tif" --i2="file2.tif" ^
-o=outfile.html
```

7.12.16 CumulativeDistribution

Converts a raster image to its cumulative distribution function.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
cumulative_distribution(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=CumulativeDistribution -v ^
--wd="/path/to/data/" -i=DEM.tif -o=output.tif
```

7.12.17 Decrement

Decreases the values of each grid cell in an input raster by 1.0 (see also InPlaceSubtract).

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
decrement(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Decrement -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.18 Divide

Performs a division operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
divide(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Divide -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.19 EqualTo

Performs a equal-to comparison operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
equal_to(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=EqualTo -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.20 Exp

Returns the exponential (base e) of values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
exp(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Exp -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.21 Exp2

Returns the exponential (base 2) of values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
exp2(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Exp2 -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.22 ExtractRasterStatistics

Extracts descriptive statistics for a group of patches in a raster.

Parameters:

Flag	Description
-i, --input	Input data raster file
--features	Input feature definition raster file
-o, --output	Output raster file
--stat	Statistic to extract
--out_table	Output HTML Table file

Python function:

```
extract_raster_statistics(
    i,
    features,
    output=None,
    stat="average",
    out_table=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ExtractRasterStatistics -v ^
--wd="/path/to/data/" -i='input.tif' --features='groups.tif' ^
-o='output.tif' --stat='minimum'
>>./whitebox_tools ^
-r=ExtractRasterStatistics -v --wd="/path/to/data/" ^
-i='input.tif' --features='groups.tif' ^
--out_table='output.html'
```

7.12.23 Floor

Returns the largest (closest to positive infinity) value that is less than or equal to the values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
floor(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Floor -v --wd="/path/to/data/" ^
-i='input.tif' -o='output.tif'
```

7.12.24 GreaterThan

Performs a greater-than comparison operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file
--incl_equals	Perform a greater-than-or-equal-to operation

Python function:

```
greater_than(
    input1,
    input2,
    output,
    incl_equals=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=GreaterThan -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif ^
--incl_equals
```

7.12.25 ImageAutocorrelation

Performs Moran's I analysis on two or more input images.

Parameters:

Flag	Description
-i, --inputs	Input raster files
--contiguity	Contiguity type
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
image_autocorrelation(
    inputs,
    output,
    contiguity="Rook",
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ImageAutocorrelation -v ^
--wd="/path/to/data/" -i="file1.tif, file2.tif, file3.tif" ^
-o=outfile.html --contiguity=Bishops
```

7.12.26 ImageCorrelation

Performs image correlation on two or more input images.

Parameters:

Flag	Description
-i, --inputs	Input raster files
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
image_correlation(
    inputs,
    output=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ImageCorrelation -v ^
--wd="/path/to/data/" -i="file1.tif, file2.tif, file3.tif" ^
-o=outfile.html
```

7.12.27 ImageRegression

Performs image regression analysis on two input images.

Parameters:

Flag	Description
--i1, --input1	Input raster file (independent variable, X)
--i2, --input2	Input raster file (dependent variable, Y)
-o, --output	Output HTML file for regression summary report
--out_residuals	Output raster regression resdidual file
--standardize	Optional flag indicating whether to standardize the residuals map

Python function:

```
image_regression(
    input1,
    input2,
    output,
    out_residuals=None,
    standardize=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ImageRegression -v ^
--wd="/path/to/data/" --i1='file1.tif' --i2='file2.tif' ^
-o='outfile.html' --out_residuals='residuals.tif' ^
--standardize
```

7.12.28 InPlaceAdd

Performs an in-place addition operation (input1 += input2).

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file or constant value

Python function:

```
in_place_add(
    input1,
    input2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=InPlaceAdd -v --wd="/path/to/data/" ^
```

```
--input1='in1.tif' --input2='in2.tif'"
>>./whitebox_tools ^
-r=InPlaceAdd -v --wd="/path/to/data/" --input1='in1.tif' ^
--input2=10.5'
```

7.12.29 InPlaceDivide

Performs an in-place division operation (input1 /= input2).

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file or constant value

Python function:

```
in_place_divide(
    input1,
    input2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=InPlaceDivide -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif'"
>>./whitebox_tools ^
-r=InPlaceDivide -v --wd="/path/to/data/" --input1='in1.tif' ^
--input2=10.5'
```

7.12.30 InPlaceMultiply

Performs an in-place multiplication operation (input1 *= input2).

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file or constant value

Python function:

```
in_place_multiply(
    input1,
    input2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=InPlaceMultiply -v ^
--wd="/path/to/data/" --input1='in1.tif' ^
--input2='in2.tif'"
>>./whitebox_tools -r=InPlaceMultiply -v ^
--wd="/path/to/data/" --input1='in1.tif' --input2=10.5'
```

7.12.31 InPlaceSubtract

Performs an in-place subtraction operation (input1 -= input2).

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file or constant value

Python function:

```
in_place_subtract(
    input1,
    input2,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=InPlaceSubtract -v ^
--wd="/path/to/data/" --input1='in1.tif' ^
--input2='in2.tif'"
>>./whitebox_tools -r=InPlaceSubtract -v ^
--wd="/path/to/data/" --input1='in1.tif' --input2=10.5'
```

7.12.32 Increment

Increases the values of each grid cell in an input raster by 1.0. (see also InPlaceAdd).

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
increment(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Increment -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.33 IntegerDivision

Performs an integer division operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
integer_division(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=IntegerDivision -v ^
--wd="/path/to/data/" --input1='in1.tif' --input2='in2.tif' ^
-o=output.tif
```

7.12.34 IsNoData

Identifies NoData valued pixels in an image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
is_no_data(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=IsNoData -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.35 KSTestForNormality

Evaluates whether the values in a raster are normally distributed.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output HTML file
--num_samples	Number of samples. Leave blank to use whole image

Python function:

```
ks_test_for_normality(
    i,
    output,
    num_samples=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=KSTestForNormality -v ^
```

```
--wd="/path/to/data/" -i=input.tif -o=output.html ^
--num_samples=1000
>>./whitebox_tools -r=KSTestForNormality -v ^
--wd="/path/to/data/" -i=input.tif -o=output.html
```

7.12.36 KappaIndex

Performs a kappa index of agreement (KIA) analysis on two categorical raster files.

Parameters:

Flag	Description
--i1, --input1	Input classification raster file
--i2, --input2	Input reference raster file
-o, --output	Output HTML file

Python function:

```
kappa_index(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=KappaIndex -v --wd="/path/to/data/" ^
--i1=class.tif --i2=reference.tif -o=kia.html
```

7.12.37 LessThan

Performs a less-than comparison operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file
--incl_equals	Perform a less-than-or-equal-to operation

Python function:

```
less_than(
    input1,
    input2,
    output,
    incl_equals=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LessThan -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif ^
--incl_equals
```

7.12.38 ListUniqueValues

Lists the unique values contained in a field witin a vector's attribute table.

Parameters:

Flag	Description
-i, --input	Input raster file
--field	Input field name in attribute table
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
list_unique_values(
    i,
    field,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ListUniqueValues -v ^
--wd="/path/to/data/" -i=lakes.shp --field=HEIGHT ^
-o=outfile.html
```

7.12.39 Ln

Returns the natural logarithm of values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
ln(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Ln -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.40 Log10

Returns the base-10 logarithm of values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
log10(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Log10 -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.41 Log2

Returns the base-2 logarithm of values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
log2(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Log2 -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.42 Max

Performs a MAX operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
max(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Max -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.43 Min

Performs a MIN operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
min(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Min -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.44 Modulo

Performs a modulo operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
modulo(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Modulo -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.45 Multiply

Performs a multiplication operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
multiply(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Multiply -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.46 Negate

Changes the sign of values in a raster or the 0-1 values of a Boolean raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
negate(
    i,
    output,
```

```
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Negate -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.47 Not

Performs a logical NOT operator on two Boolean raster images.

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file
-o, --output	Output raster file

Python function:

```
Not(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Not -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.48 NotEqualTo

Performs a not-equal-to comparison operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
not_equal_to(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=NotEqualTo -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.49 Or

Performs a logical OR operator on two Boolean raster images.

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file
-o, --output	Output raster file

Python function:

```
Or(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Or -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.50 Power

Raises the values in grid cells of one rasters, or a constant value, by values in another raster or constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
power(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Power -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.51 PrincipalComponentAnalysis

Performs a principal component analysis (PCA) on a multi-spectral dataset.

Parameters:

Flag	Description
-i, --inputs	Input raster files
--out_html	Output HTML report file
--num_comp	Number of component images to output; <= to num. input images
--standardized	Perform standardized PCA?

Python function:

```
principal_component_analysis(
    inputs,
    out_html=None,
    num_comp=None,
    standardized=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=PrincipalComponentAnalysis -v ^
--wd='/path/to/data/' -i='image1.tif;image2.tif;image3.tif' ^
--out_html=report.html --num_comp=3 --standardized
```

7.12.52 Quantiles

Transforms raster values into quantiles.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--num_quantiles	Number of quantiles

Python function:

```
quantiles(
    i,
    output,
    num_quantiles=4,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Quantiles -v --wd="/path/to/data/" ^
-i=DEM.tif -o=output.tif --num_quantiles=5
```

7.12.53 RandomField

Creates an image containing random values.

Parameters:

Flag	Description
-i, --base	Input raster file
-o, --output	Output raster file

Python function:

```
random_field(
    base,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RandomField -v --wd="/path/to/data/" ^
--base=in.tif -o=out.tif
```

7.12.54 RandomSample

Creates an image containing randomly located sample grid cells with unique IDs.

Parameters:

Flag	Description
-i, --base	Input raster file
-o, --output	Output raster file
--num_samples	Number of samples

Python function:

```
random_sample(
    base,
    output,
    num_samples=1000,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RandomSample -v --wd="/path/to/data/" ^
--base=in.tif -o=out.tif --num_samples=1000
```

7.12.55 RasterHistogram

Creates a histogram from raster values.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output HTML file (default name will be based on input file if unspecified)

Python function:

```
raster_histogram(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RasterHistogram -v ^
--wd="/path/to/data/" -i="file1.tif" -o=outfile.html
```

7.12.56 RasterSummaryStats

Measures a rasters average, standard deviation, num. non-nodata cells, and total.

Parameters:

Flag	Description
-i, --input	Input raster file

Python function:

```
raster_summary_stats(
    i,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RasterSummaryStats -v ^
--wd="/path/to/data/" -i=DEM.tif
```

7.12.57 Reciprocal

Returns the reciprocal (i.e. 1 / z) of values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
reciprocal(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Reciprocal -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.58 RescaleValueRange

Performs a min-max contrast stretch on an input greytone image.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--out_min_val	New minimum value in output image
--out_max_val	New maximum value in output image
--clip_min	Optional lower tail clip value
--clip_max	Optional upper tail clip value

Python function:

```
rescale_value_range(
    i,
    output,
    out_min_val,
    out_max_val,
    clip_min=None,
    clip_max=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RescaleValueRange -v ^
--wd="/path/to/data/" -i=input.tif -o=output.tif ^
--out_min_val=0.0 --out_max_val=1.0
>>./whitebox_tools ^
-r=RescaleValueRange -v --wd="/path/to/data/" -i=input.tif ^
-o=output.tif --out_min_val=0.0 --out_max_val=1.0 ^
--clip_min=45.0 --clip_max=200.0
```

7.12.59 RootMeanSquareError

Calculates the RMSE and other accuracy statistics.

Parameters:

Flag	Description
-i, --input	Input raster file
--base	Input base raster file used for comparison

Python function:

```
root_mean_square_error(
    i,
    base,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RootMeanSquareError -v ^
--wd="/path/to/data/" -i=DEM.tif
```

7.12.60 Round

Rounds the values in an input raster to the nearest integer value.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
round(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Round -v --wd="/path/to/data/" ^
```

```
-i='input.tif' -o=output.tif
```

7.12.61 Sin

Returns the sine (sin) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
sin(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Sin -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.62 Sinh

Returns the hyperbolic sine (sinh) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
sinh(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Sinh -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.63 Square

Squares the values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
square(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Square -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.64 SquareRoot

Returns the square root of the values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
square_root(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=SquareRoot -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.65 Subtract

Performs a differencing operation on two rasters or a raster and a constant value.

Parameters:

Flag	Description
--input1	Input raster file or constant value
--input2	Input raster file or constant value
-o, --output	Output raster file

Python function:

```
subtract(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Subtract -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.66 Tan

Returns the tangent (tan) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
tan(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Tan -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.67 Tanh

Returns the hyperbolic tangent (tanh) of each values in a raster.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
tanh(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Tanh -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.68 ToDegrees

Converts a raster from radians to degrees.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
to_degrees(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ToDegrees -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.69 ToRadians

Converts a raster from degrees to radians.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
to_radians(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ToRadians -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif
```

7.12.70 Truncate

Truncates the values in a raster to the desired number of decimal places.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file
--num_decimals	Number of decimals left after truncation (default is zero)

Python function:

```
truncate(
    i,
    output,
    num_decimals=None,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Truncate -v --wd="/path/to/data/" ^
-i='input.tif' -o=output.tif --num_decimals=2
```

7.12.71 TurningBandsSimulation

Creates an image containing random values based on a turning-bands simulation.

Parameters:

Flag	Description
-i, --base	Input base raster file
-o, --output	Output file
--range	The field's range, in xy-units, related to the extent of spatial autocorrelation
--iterations	The number of iterations

Python function:

```
turning_bands_simulation(
    base,
    output,
    range,
    iterations=1000,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TurningBandsSimulation -v ^
--wd="/path/to/data/" --base=in.tif -o=out.tif --range=850.0 ^
--iterations=2500
```

7.12.72 Xor

Performs a logical XOR operator on two Boolean raster images.

Parameters:

Flag	Description
--input1	Input raster file
--input2	Input raster file
-o, --output	Output raster file

Python function:

```
xor(
    input1,
    input2,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=Xor -v --wd="/path/to/data/" ^
--input1='in1.tif' --input2='in2.tif' -o=output.tif
```

7.12.73 ZScores

Standardizes the values in an input raster by converting to z-scores.

Parameters:

Flag	Description
-i, --input	Input raster file
-o, --output	Output raster file

Python function:

```
z_scores(
    i,
    output,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ZScores -v --wd="/path/to/data/" ^
-i=DEM.tif -o=output.tif
```

7.13 Stream Network Analysis

7.13.1 DistanceToOutlet

Calculates the distance of stream grid cells to the channel network outlet cell.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
distance_to_outlet(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=DistanceToOutlet -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=DistanceToOutlet -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.2 ExtractStreams

Extracts stream grid cells from a flow accumulation raster.

Parameters:

Flag	Description
--flow_accum	Input raster D8 flow accumulation file
-o, --output	Output raster file
--threshold	Threshold in flow accumulation values for channelization

Flag	Description
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
extract_streams(
    flow_accum,
    output,
    threshold,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ExtractStreams -v --wd="/path/to/data/" ^
--flow_accum='d8accum.tif' -o='output.tif' --threshold=100.0 ^
--zero_background
```

7.13.3 ExtractValleys

Identifies potential valley bottom grid cells based on local topolography alone.

Parameters:

Flag	Description
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--variant	Options include 'lq' (lower quartile), 'JandR' (Johnston and Rosenfeld), and 'PandD' (Peucker and Douglas); default is 'lq'
--line_thin	Optional flag indicating whether post-processing line-thinning should be performed
--filter	Optional argument (only used when variant='lq') providing the filter size, in grid cells, used for lq-filtering (default is 5)

Python function:

```
extract_valleys(
    dem,
    output,
    variant="Lower Quartile",
    line_thin=True,
    filter=5,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ExtractValleys -v --wd="/path/to/data/" ^
--dem=pointer.tif -o=out.tif --variant='JandR' ^
--line_thin
>>./whitebox_tools -r=ExtractValleys -v ^
--wd="/path/to/data/" --dem=pointer.tif -o=out.tif ^
--variant='lq' --filter=7 --line_thin
```

7.13.4 FarthestChannelHead

Calculates the distance to the furthest upstream channel head for each stream cell.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
farthest_channel_head(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FarthestChannelHead -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=FarthestChannelHead -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.5 FindMainStem

Finds the main stem, based on stream lengths, of each stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
find_main_stem(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=FindMainStem -v --wd="/path/to/data/" ^
--d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=FindMainStem -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.6 HackStreamOrder

Assigns the Hack stream order to each tributary in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
hack_stream_order(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HackStreamOrder -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=HackStreamOrder -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.7 HortonStreamOrder

Assigns the Horton stream order to each tributary in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
horton_stream_order(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=HortonStreamOrder -v ^
```

```
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=HortonStreamOrder -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.8 LengthOfUpstreamChannels

Calculates the total length of channels upstream.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
length_of_upstream_channels(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LengthOfUpstreamChannels -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=LengthOfUpstreamChannels -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.9 LongProfile

Plots the stream longitudinal profiles for one or more rivers.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
--dem	Input raster DEM file
-o, --output	Output HTML file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
long_profile(
    d8_pntr,
    streams,
    dem,
    output,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LongProfile -v --wd="/path/to/data/" ^
--d8_pntr=D8.tif --streams=streams.tif --dem=dem.tif ^
-o=output.html --esri_pntr
```

7.13.10 LongProfileFromPoints

Plots the longitudinal profiles from flow-paths initiating from a set of vector points.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--points	Input vector points file
--dem	Input raster DEM file
-o, --output	Output HTML file
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
long_profile_from_points(
    d8_pntr,
    points,
    dem,
    output,
```

```
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=LongProfileFromPoints -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --points=stream_head.shp ^
--dem=dem.tif -o=output.html --esri_pntr
```

7.13.11 RasterizeStreams

Rasterizes vector streams based on Lindsay (2016) method.

Parameters:

Flag	Description
--streams	Input vector streams file
--base	Input base raster file
-o, --output	Output raster file
--nodata	Use NoData value for background?
--feature_id	Use feature number as output value?

Python function:

```
rasterize_streams(
    streams,
    base,
    output,
    nodata=True,
    feature_id=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RasterizeStreams -v ^
--wd="/path/to/data/" --streams=streams.shp --base=raster.tif ^
-o=output.tif
```

7.13.12 RemoveShortStreams

Removes short first-order streams from a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--min_length	Minimum tributary length (in map units) used for network prunning
--esri_pntr	D8 pointer uses the ESRI style scheme

Python function:

```
remove_short_streams(
    d8_pntr,
    streams,
    output,
    min_length,
    esri_pntr=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=RemoveShortStreams -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
```

7.13.13 ShreveStreamMagnitude

Assigns the Shreve stream magnitude to each link in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
shreve_stream_magnitude(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
```

```
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=ShreveStreamMagnitude -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=ShreveStreamMagnitude -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.14 StrahlerStreamOrder

Assigns the Strahler stream order to each link in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
strahler_stream_order(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StrahlerStreamOrder -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=StrahlerStreamOrder -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.15 StreamLinkClass

Identifies the exterior/interior links and nodes in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
stream_link_class(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StreamLinkClass -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=StreamLinkClass -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.16 StreamLinkIdentifier

Assigns a unique identifier to each link in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
stream_link_identifier(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StreamLinkIdentifier -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=StreamLinkIdentifier -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.17 StreamLinkLength

Estimates the length of each link (or tributary) in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--linkid	Input raster streams link ID (or tributary ID) file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
stream_link_length(
    d8_pntr,
    linkid,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StreamLinkLength -v ^
```

```
--wd="/path/to/data/" --d8_pntr=D8.tif --linkid=streamsID.tif ^
--dem=dem.tif -o=output.tif
>>./whitebox_tools ^
-r=StreamLinkLength -v --wd="/path/to/data/" --d8_pntr=D8.tif ^
--linkid=streamsID.tif --dem=dem.tif -o=output.tif --esri_pntr ^
--zero_background
```

7.13.18 StreamLinkSlope

Estimates the average slope of each link (or tributary) in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--linkid	Input raster streams link ID (or tributary ID) file
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
stream_link_slope(
    d8_pntr,
    linkid,
    dem,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StreamLinkSlope -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --linkid=streamsID.tif ^
--dem=dem.tif -o=output.tif
>>./whitebox_tools ^
-r=StreamLinkSlope -v --wd="/path/to/data/" --d8_pntr=D8.tif ^
--linkid=streamsID.tif --dem=dem.tif -o=output.tif --esri_pntr ^
--zero_background
```

7.13.19 StreamSlopeContinuous

Estimates the slope of each grid cell in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-i, --dem	Input raster DEM file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
stream_slope_continuous(
    d8_pntr,
    streams,
    dem,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=StreamSlopeContinuous -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --linkid=streamsID.tif ^
--dem=dem.tif -o=output.tif
>>./whitebox_tools ^
-r=StreamSlopeContinuous -v --wd="/path/to/data/" ^
--d8_pntr=D8.tif --streams=streamsID.tif --dem=dem.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.20 TopologicalStreamOrder

Assigns each link in a stream network its topological order.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file

Flag	Description
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
topological_stream_order(
    d8_pntr,
    streams,
    output,
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TopologicalStreamOrder -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=TopologicalStreamOrder -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

7.13.21 TributaryIdentifier

Assigns a unique identifier to each tributary in a stream network.

Parameters:

Flag	Description
--d8_pntr	Input raster D8 pointer file
--streams	Input raster streams file
-o, --output	Output raster file
--esri_pntr	D8 pointer uses the ESRI style scheme
--zero_background	Flag indicating whether a background value of zero should be used

Python function:

```
tributary_identifier(
    d8_pntr,
    streams,
    output,
```

```
    esri_pntr=False,
    zero_background=False,
    callback=default_callback)
```

Command-line Interface:

```
>>./whitebox_tools -r=TributaryIdentifier -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif
>>./whitebox_tools -r=TributaryIdentifier -v ^
--wd="/path/to/data/" --d8_pntr=D8.tif --streams=streams.tif ^
-o=output.tif --esri_pntr --zero_background
```

8. Contributing

If you would like to contribute to the project as a developer, follow these instructions to get started:

1. Fork the larger Whitebox project (in which whitebox-tools exists) (https://github.com/jblindsay/whitebox-geospatial-analysis-tools)
2. Create your feature branch (git checkout -b my-new-feature)
3. Commit your changes (git commit -am 'Add some feature')
4. Push to the branch (git push origin my-new-feature)
5. Create a new Pull Request

Unless explicitly stated otherwise, any contribution intentionally submitted for inclusion in the work shall be licensed as above without any additional terms or conditions.

If you would like to contribute financial support for the project, please contact John Lindsay. We also welcome contributions in the form of media exposure. If you have written an article or blog about *WhiteboxTools* please let us know about it.

9. Reporting Bugs

WhiteboxTools is distributed as is and without warranty of suitability for application. If you encounter flaws with the software (i.e. bugs) please report the issue. Providing a detailed description of the conditions under which the bug occurred will help to identify the bug. *Use the Issues tracker on GitHub to report issues with the software and to request feature enchancements.* Please do not email Dr. Lindsay directly with bugs.

10. Known Issues and Limitations

- There is limited support for reading, writing, or analyzing vector data yet. Plans include native support for the ESRI Shapefile format and possibly GeoJSON data.
- The LAZ compressed LiDAR data format is currently unsupported although zipped LAS files (.zip) are.
- There is no support for reading waveform data contained within or associated with LAS files.
- File directories cannot contain apostrophes (', e.g. /John's data/) as they will be interpreted in the arguments array as single quoted strings.
- The Python scripts included with **WhiteboxTools** require Python 3. They will not work with Python 2, which is frequently the default Python version installed on many systems.

11. License

The **WhiteboxTools** library is distributed under the MIT license, a permissive open-source (free software) license.

12. Frequently Asked Questions

12.1 Do I need Whitebox GAT to use WhiteboxTools?

No you do not. You can call the tools contained within *WhiteboxTools* completely independent from the *Whitebox GAT* user interface using a Remote Procedure Call (RPC) approach. In fact, you can interact with the tools using Python scripting or directly, using a terminal application (command prompt). See *Interacting With* WhiteboxTools* From the Command Prompt* for further details.

12.2 How do I request a tool be added?

Eventually most of the tools in *Whitebox GAT* will be ported over to *WhiteboxTools* and all new tools will be added to this library as well. Naturally, this will take time. The order by which tools are ported is partly a function of ease of porting, existing infrastructure (i.e. raster and LiDAR tools will be ported first since their is currently no support in the library for vector I/O), and interest. If you are interested in making a tool a higher priority for porting, email John Lindsay.

12.3 Can WhiteboxTools be incorporated into other software and open-source GIS projects?

WhiteboxTools was developed with the open-source GIS Whitebox GAT in mind. That said, the tools can be accessed independently and so long as you abide by the terms of the MIT license, there is no reason why other software and GIS projects cannot use *WhiteboxTools* as well. In fact, this was one of the motivating factors for creating the library in the first place. Feel free to use *WhiteboxTools* as the geospatial analysis engine in your open-source software project.

12.4 What platforms does WhiteboxTools support?

WhiteboxTools is developed using the Rust programming language, which supports a wide variety of platforms including MS Windows, MacOS, and Linux operating systems and common chip architectures. Interestingly, Rust also supports mobile platforms, and *WhiteboxTools* should therefore be capable of targeting (although no testing has been completed in this regard to date). Nearly all development and testing of the software is currently carried out on MacOS and we cannot guarantee a bug-free performance on other platforms. In particularly, MS Windows is the most different from the other platforms and is therefore the most likely to encounter platform-specific bugs. If you encounter bugs in the software, please consider reporting an issue using the GitHub support for issue-tracking.

12.5 What are the system requirements?

The answer to this question depends strongly on the type of analysis and data that you intend to process. However, generally we find performance to be optimal with a recommended minimum of 8-16GB of memory (RAM), a modern multi-core processor (e.g. 64-bit i5 or i7), and an solid-state-drive (SSD). It is likely that *WhiteboxTools* will have satisfactory performance on lower-spec systems if smaller datasets are being processed. Because *WhiteboxTools* reads entire raster datasets into system memory (for optimal performance, and in recognition that modern systems have increasingly larger amounts of fast RAM), this tends to be the limiting factor for the upper-end of data size successfully processed by the library. 64-bit operating systems are recommended and extensive testing has not been carried out on 32-bit OSs. See **"What platforms does WhiteboxTools support?"** for further details on supported platforms.

12.6 Are pre-compiled executables of WhiteboxTools available?

Pre-compiled binaries for *WhiteboxTools* can be downloaded from the *Geomorphometry and Hydrogeomatics Research Group* software web site for various supported operating systems. If you need binaries for other operating systems/system architectures, you will need to compile the executable from source files. See Installation for details.

12.7 Why is WhiteboxTools programmed in Rust?

I spent a long time evaluating potential programming language for future development efforts for the *Whitebox GAT* project. My most important criterion for a language was that it compile to native code, rather than target the Java virtual machine (JVM). I have been keen to move Whitebox GAT away from Java because of some of the challenges that supporting the JVM has included for many Whitebox users. The language should be fast and productive–Java is already quite fast, but if I am going to change development languages, I would like a performance boost. Furthermore, given that many, though not all, of the algorithms used for geospatial analysis scale well with concurrent (parallel) implementations, I favoured languages that offered easy and safe concurrent programming. Although many would consider C/C++ for this work, I was looking for a modern and safe language. Fortunately, we are living through a renaissance period in programming language development and there are many newer languages that fit the bill nicely. Over the past two years, I considered each of Go, Rust, D, Nim, and Crystal for Whitebox development and ultimately decided on Rust. [See *GoSpatial* and *lidario*.]

Each of the languages I examined has its own advantages of disadvantages, so why Rust? It's a combination of factors that made it a compelling option for this project. Compared with many on the list, Rust is a mature language with a vibrant user community. Like C/C++, it's a high-performance and low-level language that allows for complete control of the system. However, Rust is also one of the safest languages, meaning that I can be confident that *WhiteboxTools* will not contain common bugs, such as memory use-after-release, memory leaks and race conditions within concurrent code. Importantly, and quite uniquely, this safety is achieved in the Rust language without the use of a garbage collector (automatic memory management). Garbage collectors can be great, but they do generally come with a certain efficiency trade-off that Rust does not have. The other main advantage of Rust's approach to memory management is that it allows for a level of interaction with scripting languages (e.g. Python) that is quite difficult to do in garbage collected languages. Although **WhiteboxTools** is currently set up to use an automation approach to interacting with Python code that calls it, I like the fact that I have the option to create a *WhiteboxTools* shared library.

Not everything with Rust is perfect however. It is still a very young language and there are many pieces still missing from its ecosystem. Furthermore, it is not the easiest language to learn, particularly for people who are inexperienced with programming. This may limit my ability to attract other programers to the Whitebox project, which would be unfortunate. However, overall, Rust was the best option for this particular application.

12.8 Do I need Rust installed on my computer to run WhiteboxTools?

No, you would only need Rust installed if you were compiling the *WhiteboxTools* codebase from source files.

12.9 How does WhiteboxTools' design philosophy differ?

Whitebox GAT is frequently praised for its consistent design and ease of use. Like *Whitebox GAT*, *WhiteboxTools* follows the convention of *one tool for one function*. For example, in *WhiteboxTools* assigning the links in a stream channel network their Horton, Strahler, Shreve, or Hack stream ordering numbers requires running separate tools (i.e. *HortonStreamOrder*, *StrahlerStreamOrder*, *ShreveStreamMagnitude*, and *HackStreamOrder*). By contrast, in GRASS GIS[1] and ArcGIS single tools (i.e. the *r.stream.order* and *Stream Order* tools respectively) can be configured to output different channel ordering schemes. The *WhiteboxTools* design is intended to simplify the user experience and to make it easier to find the right tool for a task. With more specific tool names that are reflective of their specific purposes, users are not as reliant on reading help documentation to identify the tool for the task at hand. Similarly, it is not uncommon for tools in other GIS to have multiple outputs. For example, in GRASS GIS the *r.slope.aspect* tool can be configured to output slope, aspect, profile curvature, plan curvature, and several other common terrain surface derivatives. Based on the *one tool for one function* design approach of *WhiteboxTools*, multiple outputs are indicative that a tool should be split into different, more specific tools. Are you more likely to go to a tool named *r.slope.aspect* or *TangentialCurvature* when you want to create a tangential curvature raster from a DEM? If you're new to the software and are unfamiliar with it, probably the later is more obvious. The *WhiteboxTools* design approach also has the added benefit of simplifying the documentation for tools. The one downside to this design approach, however, is that it results (or will result) in a large number of tools, often with signifcant overlap in function.

[1] NOTE: It's not my intent to criticize GRASS GIS, as I deeply respect the work that the GRASS developers have contributed. Rather, I am contrasting the consequences of *WhiteboxTools'* design philosophy to that of other GIS.