Using the Open-source TopoFlow Python Package for Extracting D8-based Grids from DEMs and for Fluvial Landscape Evolution Modeling

Scott D. Peckham
University of Colorado
Boulder, Colorado USA
Scott.Peckham@colorado.edu

***Abstract*—The TopoFlow 3.5 Python package consists of over 72,000 lines of code and provides a full-featured, spatial hydrologic model with several alternate and easily swappable process components for each hydrologic process (e.g. infiltration, evaporation, snowmelt). In support of its D8-based flow routing and the included landscape evolution model, Erode, this Python package also contains a rich collection of lesser-known tools for working with and extracting other gridded products from DEMs. This paper briefly describes some of these DEM-related tools and demonstrates how to use them.**

I. Introduction

TopoFlow has been under development since 2000, with the original version (*Peckham*, 2009a) written in Interactive Data Language (IDL). Over the years, TopoFlow has been completely rewritten in Python using the NumPy (Numerical Python) package, and has helped to drive advances in model coupling technologies, such as BMI, the Basic Model Interface (*Peckham et al.*, 2013), the CSDMS Standard Names (*Peckham*, 2014b) (which evolved into the Geoscience Standard Names, geostandardnames.org) and a model-coupling framework called EMELI (*Peckham*, 2014a; *Jiang et al.*, 2017).

TopoFlow 3.5 (*Peckham*, 2017a,b; *Peckham et al*, 2017) is an open-source, Python/NumPy package for spatial hydrologic modeling that contains: (1) many plug-and-play components for modeling hydrologic processes (e.g. infiltration, evaporation, channel flow, snowmelt, etc.), (2) a fluvial landscape evolution model based on D8 flow routing called *Erode*, (3) a large collection of low-level utilities (e.g. for reading and writing files) (4) example data sets and (5) a self-contained, plug-and-play model coupling framework called EMELI. Each model component has a Basic Model Interface (BMI) that uses the CSDMS Standard Names. In support of TopoFlow's D8-based flow routing and the landscape evolution model, Erode, this package contains a collection of lesser-known tools for working with DEMs, such as: (1) extracting D8-based grids from a DEM, (2) "profile smoothing" a DEM prior to using the kinematic wave flow routing method and (3) reading data from different file formats (e.g. RiverTools, NetCDF, BOV). This paper briefly describes some of the foundational, DEM-

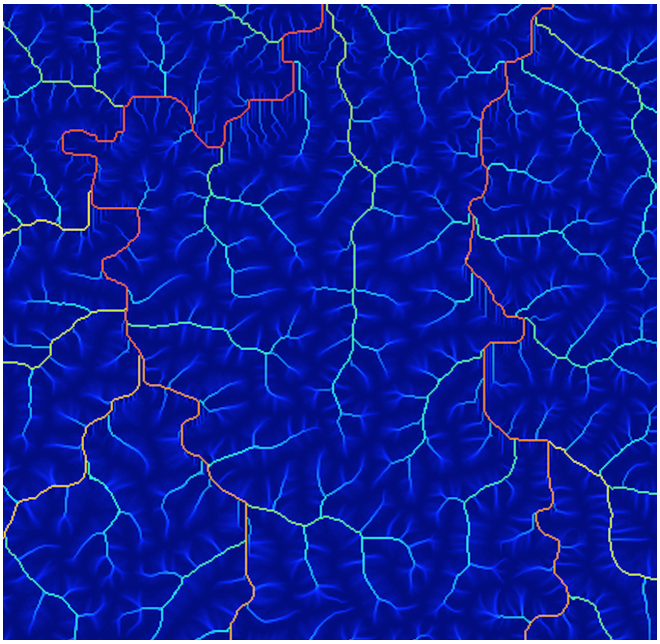

Fig. 1. Part of a total contributing area grid, Beaver Creek, Kentucky.

related tools in TopoFlow and demonstrates how to use them.

II. Working with Binary Grid Files

In the TopoFlow 3.5 Python package, it is generally assumed that DEMs and other spatial grids are stored as IEEE binary, floating-point (4-byte or 8-byte) values, in row-major order (first line is north edge) and with georeferencing information stored in a separate text file called a *header file*. This same, basic and very efficient file format is used with many different filename extensions, such as RTG (RiverTools Grid), FLT (Arc GridFloat), IMG (ENVI Grid), BOV (Brick of Values) and others. However, each of these (identical) formats uses a differently-formatted header file. By default, the TF 3.5 package assumes that the header file is stored as a RiverTools Information (RTI) file. RiverTools 4.1 is a point-and-click commercial package for terrain and hydrologic analysis also developed by the author — see *Peckham* (2009b); *Rivix* (2014). The TF package therefore includes utilities with a convenient API for working with RTG, RTI and even RTS (RiverTools Sequence) files (i.e. a grid stack or sequence of binary grids in one file). Modules called *rtg_files.py*, *rti_files.py* and *rts_files.py* are located in the package folder *topoflow/utils*. The package also has APIs for working with several other file formats, including: BOV, multi-column text (e.g. time series), CFG (configuration) and NetCDF. The NetCDF utilities are in four separate files for working with time series, profile series, grid series and "cube series" data types. Modules called *bov_files.py*, *cfg_files.py*, *ncts_files.py*, *ncps_files.py*, *ncgs_files.py*, *nccs_files.py* and *text_ts_files.py* (multi-column text) are also located in the package folder *topoflow/utils*. Examples of using the RTG and RTI file utilities are given in subsequent sections.

III. Filling Depressions in a DEM

Wang and Liu (2006) introduced an efficient algorithm for filling depressions in a DEM that is based on the computer science concept of a *priority queue*. The TopoFlow 3.5 Python package contains an implementation of this algorithm that can use either (1) the built-in Python package *heapq* or (2) an alternate, pure Python implementation of the binary, heap-based priority queue. The depression-filling implementation and alternate priority queue modules are called: *fill_pits.py* and *heap_base.py* and are in the package folder *topoflow/utils*. The following lines of code illustrate how this tool can be used directly. However, it is also called from the *d8_base.py* module (in the *topoflow/components* folder), when the FILL_PITS_IN_Z0 flag is set in the component's configuration file.

```
from topoflow.utils import fill_pits
from topoflow.utils import rtg_files
from topoflow.utils import rti_files
in_dir = 'Users/peckham/TF_Tests/Basin1_runs/'
in_prefix = in_dir + 'Basin1'
DEM_file = in_prefix + '_rawdem.rtg'
header_file = in_prefix + '.rti'
grid_info = rti_files.read_info( header_file,
   REPORT=True)
DEM = rtg_files.read_grid( DEM_file, grid_info,
   SILENT=False )
fill_pits.fill_pits( DEM, 'FLOAT',
   grid_info.ncols, grid_info.nrows, SILENT=False)
# Save new DEM to a file
new_DEM_file = in_prefix + '_dem.rtg'
rtg_files.write_grid( DEM, new_DEM_file,
   grid_info)
```

IV. Extracting D8-based Grids from a DEM

The TopoFlow 3.5 Python package contains a powerful D8 toolkit which is able to compute many different types of grids associated with the D8 flow routing method (*Gruber and Peckham*, 2009; *Jenson*, 1985). This includes grids of D8 flow direction codes, total and specific contributing area, local topographic slope and others. The corresponding modules in the *topoflow/components* folder are: *d8_base.py*, *d8_global.py* and *d8_local.py*. The last two of these inherit for the first one, and the last one is only for use with Erode D8 Local, described later. Like all other TopoFlow components, these can be configured by editing a configuration file (e.g. with filename ending in *_d8_global.cfg*.)

The following commands show how to use TopoFlow?s D8 toolkit to compute (1) a D8 flow direction grid (with *Jenson* (1985) flow codes), (2) a D8 topographic slope grid, and (3) a D8 total contributing area grid. All of these grids have the same dimensions as the source DEM they are derived from. TopoFlow uses two types of filename prefix to help organize files — a **site prefix** is

used for files that describe the study site and therefore don't change between model runs (e.g. the DEM), while a **case prefix** is used for files that result from running the model for a particular scenario or case (e.g. response to a given storm or component set). Before running this code, you must create a directory in your home directory, e.g. "/Users/peckham/TF_Tests/Basin1_runs" and copy a DEM as a binary grid along with an RTI header file into the new directory as well as a CFG file for the D8 Global component with extension "_d8_global.cfg". Then cd to this directory and use it for both in_directory and out_directory in the following. Allowing these two directories to be different provides maximum flexibility, e.g. several users can share data in the same input directory but save results in their own output directory, and different components can use different output directories.

```
import topoflow
from topoflow.components import d8_global
d8 = d8_global.d8_component()
d8.DEBUG = False
in_dir = '/Users/peckham/TF_Tests/Basin1_runs/'
site_prefix = 'Basin1'
filename = site_prefix + '_d8_global.cfg'
# Construct full path to configuration file
cfg_file = in_dir + filename
time = 0.0
d8.initialize( cfg_file=cfg_file,
   SILENT=False, REPORT=True )
d8.update( time, SILENT=False, REPORT=True )
```

Once the above set of D8-based grids have been computed for a given DEM, they can be saved into binary grid files (IEEE binary, row-major, 4-byte floats) with the following commands.

```
from topoflow.utils import rtg_files
from topoflow.utils import rti_files
out_dir = '/Users/peckham/TF_Tests/Basin1_runs/'
out_prefix = out_dir + site_prefix
header_file = (out_prefix + '.rti')
grid_info = rti_files.read_info( header_file,
   REPORT=True)

# Save D8 flow code grid
d8_code_file = (out_prefix + '_flow.rtg')
rtg_files.write_grid( d8.d8_grid, d8_code_file,
   grid_info, RTG_type='BYTE')

# Save D8 contributing area grid
d8_area_file = (out_prefix + '_d8-area.rtg')
rtg_files.write_grid( d8.A, d8_area_file,
   grid_info, RTG_type='FLOAT')

# Compute the D8 slope grid
pIDs = d8.parent_IDs
d8_slope = (d8.DEM - d8.DEM[ pIDs ]) / d8.ds

# Save the D8 slope grid
d8_slope_file = (out_prefix + '_d8-slope.rtg')
rtg_files.write_grid( d8_slope, d8_slope_file,
   grid_info, RTG_type='FLOAT')
```

Every DEM grid cell has one *D8 parent* cell that it flows towards, but a D8 parent can have multiple *D8 kids*. Each cell has an ID, sometimes a long integer (calendar-style numbering) and sometimes a (row, col) tuple. The Erode D8 Global component for landscape evolution modeling (discussed later) generates a sequence of DEMs indexed by time. After each time step, it calls the D8 Global component on the modified DEM to rapidly update the D8 grids. The Erode D8 Global source code therefore also illustrates how to call the D8 Global component.

V. DEM Profile Smoother Tool

This is a pre-processing tool that can be applied to a DEM to create a new DEM with smoother and more realistic channel slopes. Well-defined and smoothly-varying slopes along channel streamlines is important when using the kinematic wave method of flow routing. TopoFlow's kinematic wave flow routing component, *channels_kinematic_wave.py* is in the *topoflow/components* folder. The algorithm, based on Flint's Law, first computes a new D8 slope grid (topographic slopes) from a D8 area grid and then computes a new DEM from the D8 slope grid. For more details, see *Peckham* (2009c). This module, called *smooth_DEM.py*, is in the *topoflow/components* folder.

VI. Fluvial Landscape Evolution Modeling

Erode is a fluvial landscape evolution model (LEM) that is included as a component in the TopoFlow 3.5 Python package. For background on fluvial landscape evolution modeling, see *Peckham* (2003). It has a BMI interface and can be run by itself or using EMELI. Unlike most (or perhaps all) other LEMs, Erode does not fill pits in the

initial DEM artificially at the start. Instead, local depressions are filled naturally by the sediment transport process itself over time. Movies of the D8 area grid evolving over time show what looks like channel avulsions (i.e. entire channels sometimes "jumping" to a new pathway) during this filling process. In addition, Erode includes a robust numerical stability condition and uses adaptive time stepping for optimum performance. Erode is not typically coupled to the hydrologic model components in TopoFlow, but could easily be modified to provide a sediment or contaminant transport model component that could be coupled to the channel flow components. Within the TopoFlow 3.5 package, the main version of Erode (Erode D8 Global) is the module in the *topoflow/components* folder called *erode_d8_global.py*, which inherits from the base class module *erode_base.py*. Another, experimental and less robust version of Erode (Erode D8 Local) is also included that uses *local* time stepping. That is, it uses a *discrete event simulation* (DES) algorithm for solving stiff partial differential equations, in which every grid cell can have a different (local) time step, as opposed to all grid cells using the same time step (global). This module, called *erode_d8_local.py*, also inherits from *erode_base.py* and uses the *d8_local.py* module.

VII. Summary

The TopoFlow 3.5 Python package has a rich, object-oriented collection of open-source utilities and components, some of which perform common DEM processing and extraction tasks. Package installation instructions are given in an appendix to *Peckham et al* (2017). Several lesser-known capabilities of the package were highlighted, with example code snippets.

References

Gruber, S., and S. Peckham (2009) Land surface parameters and objects in hydrology, In: *Geomorphometry: Concepts, Software, Applications*, edited by T. Hengl and H. I. Reuter, pp. 171–194, Elsevier, Amsterdam, http://dx.doi.org/10.1016/S0166-2481(08)00007-X.

Jenson, S. K. (1985) Automated derivation of hydrologic basin characteristics from digital elevation model data, In: *Proceedings of the Digital Representations of Spatial Knowledge*, pp. 301–310, Auto-Carto VII, Washington, D.C., http://mapcontext.com/autocarto/proceedings/auto-carto-7/.

Jiang, P., M. Elag, P. Kumar, S.D. Peckham, L. Marini, R. Liu (2017) A service-oriented architecture for coupling web service models using the Basic Model Interface (BMI), *Environmental Modelling & Software*, **92**, 107-118, http://dx.doi.org/10.1016/j.envsoft.2017.01.021.

Peckham, S.D. (2003) Fluvial landscape models and catchment-scale sediment transport, *Global and Planetary Change* (special issue), **39**(1), 31–51, http://dx.doi.org/10.1016/S0921-8181(03)00014-6.

Peckham, S.D. (2009a) Geomorphometry and spatial hydrologic modeling, In: Hengl, T. and Reuter, H.I. (Eds), *Geomorphometry: Concepts, Software and Applications, Chapter 25*, vol. 33, Elsevier, 579–602, http://dx.doi.org/10.1016/S0166-2481(08)00025-1.

Peckham , S.D. (2009b) Geomorphometry in RiverTools, In: Hengl, T. and Reuter, H.I. (Eds), *Geomorphometry: Concepts, Software and Applications, Chapter 18*, vol. 33, Elsevier, 411–430, http://dx.doi.org/10.1016/S0166-2481(08)00018-4.

Peckham, S.D. (2009c) A new algorithm for creating DEMs with smooth elevation profiles, Proc. of Geomorphometry 2009, Zurich, Switzerland, 31 Aug. to 2 Sept., 34-37. http://geomorphometry.org/Peckham2009.

Peckham, S.D., E.W.H. Hutton and B. Norris (2013) A component-based approach to integrated modeling in the geosciences: The Design of CSDMS, Computers & Geosciences, special issue: Modeling for Environmental Change, **53**, 3–12, http://dx.doi.org/10.1016/j.cageo.2012.04.002.

Peckham, S.D. (2014a) EMELI 1.0: An experimental smart modeling framework for automatic coupling of self-describing models, Proceedings of HIC 2014, 11th International Conf. on Hydroinformatics, New York, NY. CUNY Academic Works, http://academicworks.cuny.edu/cc_conf_hic/464/.

Peckham, S.D. (2014b) The CSDMS Standard Names: Cross-domain naming conventions for describing process models, data sets and their associated variables, *Proceedings of the 7th Intl. Congress on Env. Modelling and Software*, International Environmental Modelling and Software Society (iEMSs), San Diego, CA. (Eds. D.P. Ames, N.W.T. Quinn, A.E. Rizzoli), Paper 12, http://scholarsarchive.byu.edu/iemssconference/2014/Stream-A/12/.

Peckham, S.D., M. Stoica, E.E. Jafarov, A. Endalamaw and W.R. Bolton (2017) Reproducible, component-based modeling with TopoFlow, a spatial hydrologic modeling toolkit, *Earth and Space Science*,**4**(6), 377–394, special isssue: Geoscience Papers of the Future, American Geophysical Union, http://dx.doi.org/10.1002/2016EA000237.

Peckham, S.D. (2017a) TopoFlow 3.5 Python Package, peckhams/topoflow, http://doi.org/10.5281/zenodo.322649, A spatial hydrologic model written in Python, with 16 process model components, numerous utilities, sample data, etc. All components have a BMI interface. Includes the EMELI 1.0, plug-and-play modeling framework.

Peckham, S.D. (2017b) TopoFlow Python Package on GitHub, open-source, https://github.com/peckhams/topoflow.

Rivix, LLC (2014) RiverTools 4.0 User's Guide, 250 pp.

Wang, L. and H. Liu (2006) An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modeling, *International Journal of Geographic Information Science*, **20**(2), 193–213, Taylor & Francis, http://doi.org/10.1080/13658810500433453.