Read the Docs Template Documentation

Release 3.0

Read the Docs

Jul 16, 2023

CONTENTS

CHAPTER

ONE

TABLE OF CONTENTS

1.1 What is HexWatershed?

1.1.1 Overview

HexWatershed is a mesh independent flow direction model for hydrological models.

1.1.2 Why do we develop HexWatershed

Spatial discretization is the cornerstone of all spatially-distributed numerical simulations including watershed hydrology. Traditional square grid spatial discretization has several limitations:

- It cannot represent adjacency uniformly;
- It leads to the "island effect" and the diagonal travel path issue in D8 scheme;
- It cannot provide a spherical coverage without significant spatial distortion;
- It cannot be coupled with other unstructured mesh-based models such as the oceanic models.

Therefore, we developed a watershed delineation model (HexWatershed) based on the hexagon mesh spatial discretization.

We further improve HexWatershed to fully unstructured mesh-based to support variable-resolution meshes such as the MPAS mesh.

We propose that spatially distributed hydrologic simulations should consider using a hexagon grid spatial discretization.

1.2 Installation

1.2.1 Overview

HexWatershed was originally released in C++ 11.

Since HexWatershed v3.0, it was redesigned using a **hybrid** Python (frontend) and C++ (backend) approach. Only the HexWatershed v3.0 will be maintained and supported.

The Python frontend is used to generate the mesh, build the topological relationship, and a list of other pre and post-processing algorithms.

The C++ backend is used to run the core HexWatershed model.

1.2.2 Requirements

It is possible to build the HexWatershed on Linux/Mac/Windows if your system can support both the C++ and Python environments:

C++ Requirements

CMake

cmake (v3.1.0 and above) is used to generate the makefile.

- Linux: cmake can be installed through a package manager. If you are on a Linux HPC, you administration should have already installed in most cases.
- Mac: cmake can be installed through Homebrew (https://brew.sh/): https://formulae.brew.sh/formula/cmake/
- Windows: cmake can be installed following the instruction from hexwatershed (https://cmake.org/download/).

GCC

GCC (v8.1.0 and above) is used to compile the C++ source code.

Similar to cmake, you can install GCC on linux or Mac.

- Linux: gcc can be installed through a package manager.
- Mac: gcc can be installed through Homebrew (https://formulae.brew.sh/formula/gcc).
- Windows: gcc can be installed using either Cygwin or MinGW (https://gcc.gnu.org/install/binaries.html).

A Linux version CMakeLists file is provided as an example. Two optional bash scripts are provided to assist this process.

Python requirements

Conda

The **Conda** platform is recommended to install the dependency Python package. Conda can be installed throught either https://docs.conda.io/en/latest/miniconda.html or https://anaconda.org/.

Python

Python (3.8 and above) is required to run the Python package. It can be installed using the conda tool.

Pip

The Python **pip** is used to install the HexWatershed Python package. Pip is installed along with the Python.

Step by step instruction

Only an exmaple on Mac is provided below:

- Open the terminal, cd to a desired directory and create a folder for this project:

```
mkdir hexwatershed_tutorial

cd hexwatershed_tutorial
```

- Clone the repository into the current folder and go inside it:

```
git clone https://github.com/changliao1025/hexwatershed.git

cd hexwatershed
```

- Go into the build folder and run cmake:

```
cd build

cmake CMakeLists.txt -DCMAKE_CXX_COMPILER=g++-10
```

- Build and install the HexWatershed C++ program:

```
make install
```

- Create the conda environment for the HexWatershed and activate it:

```
conda config --set channel_priority strict

conda create --name hexwatershed_tutorial python=3.8

conda activate hexwatershed_tutorial
```

- Install the package through the conda-forge channel

```
conda install -c conda-forge hexwatershed
```

Because the **GDAL** library is used by this project and the **proj** library is often not configured correctly automatically.

On Linux or Mac, you can set it up like this, .bash_profile as an example:

Anaconda:

```
export PROJ_LIB=/people/user/.conda/envs/hexwatershed_tutorial/share/proj
```

Miniconda:

```
export PROJ_LIB=/opt/miniconda3/envs/hexwatershed_tutorial/share/proj
```

By now, your should have installed both the C++ and Python components of the HexWatershed model.

Next, you can test the model with the example following this instruction. https://github.com/changliao1025/pyhexwatershed/

1.3 Application

1.3.1 Overview

The recommended approach to run a HexWatershed simulation is through the Python package interface.

1.3.2 Data structure

HexWatershed uses the JavaScript Object Notation (JSON) file format for model configuration and data exchange.

The input data includes:

- A ESRI shapefile that defines the original river network
- A raster Geotiff file that contains the digital elevation model (DEM) data

Note that depending on the configuration, not all the input files are needed, or additional input files are needed.

1.3.3 Data preparation

Because the core algorithms within HexWatershed assume that all the data are on the Geographic Coordinate System (GCS), most input and output data use GCS.

However, since most DEM data use the Projected Coordinate System (PCS), a reprojection is sometimes required.

Besides, depending on the simulation configuration, different data are needed. Below are some instructions for different scenarios.

To support all the major computer systems, we use the QGIS to operate on most spatial datasets.

Single watershed

River network

The river network file can be defined using a vector-based river flowline file.

Because the real-worlkd river network is often complex, some simplication is recommended. For example, the river network should only include major flowlines.

The shapefile should use the GCS system. If the vector you have is in a different PCS, you can re-project it to the GCS.

Boundary with buffer

This boundary with buffer zone is mainly used to extract the DEM. * obtain a vector watershed boundary (PCS system), if the boudnary is in a GCS system, you should convert it to PCS simialr to the flowline. * create a buffer zone watershed boundary (PCS system), the buffer increase distance should be linked to your resolution of intestes. For example, if the highest mesh resolution you will use is arounf 5km, then the buffer zone distance should be set to 5km.

DEM

The DEM file can be extracted from a large DEM which contains the study domain.

To do so, the recommended steps are:

- prepare a large DEM with includes the study domain (PCS system)
- overlay the DEM (PCS system) and river network, and edit the boundary (PCS system) near the outlet so that there is less extended DEM near the outlet (GCS or PCS system)
- extract the large DEM using the edited boundary (PCS system)

Continuous domain with multiple watersheds

River network

The river network file can be defined using a vector-based river flowline file.

Because the real-worlkd river network is often complex, some simplication is recommended. For example, the river network should only include major flowlines.

DEM

The DEM file can be extracted from a large DEM which contains the study domain.

Global (Disontinuous domain with multiple watersheds)

River network

Global scale hydrology dataset such as hydroshed may be used. A subset of global river network can also be used.

DEM

It is recommended to assign/inject elevation within the mesh similart to MPAS.

Table 1: Input usage by domain

Domain	Single watershed	Multiple continuous watershed	Multiple discontinuous watershed (global)
Flowline	Yes if iFlag_flowline == 1	Yes if iFlag_flowline == 1	Yes if iFlag_flowline == 1
Raster DEM	Yes if mesh type is not MPAS	Yes if mesh type is not MPAS	Yes if mesh type is not MPAS
Boudnary	Yes, it will be used to generate mesh	Yes, it will be used to generate mesh	No, so far global mesh is pre-defined

1.3.4 Model configuration

HexWatershed (as well as its submodule **PyFlowline**) uses two **JSON** files (main and basin) as the configuration files.

Main configuration

The main configuration JSON file contains the domain scale parameters. Each domain may contain one or more basins.

Table 2: Main configuration JSON file

Keyword	Data type	Value	Description
sFile-name_model_configuration	string		The full path to the main configuration file
sModel	string	"hexwatershed"	The model name
sRegion	string		The study region
sWorkspace_bin	string		The directory of the complied HexWatershed binary
sWorkspace_input	string		The directory of the input
sWorkspace_output	string		The directory of the output
sJob	string		The job name for HPC
iFlag_create_mesh	int	0 or 1	Flag to generate mesh
iFlag_save_mesh	int	0 or 1	Flag to save mesh
iFlag_simplification	int	0 or 1	Flag for flowline simplication
iFlag_intersect	int	0 or 1	Flag to flowline mesh intersection
iFlag_resample_method	int	1 or 2	Method for DEM resampling
iFlag_flowline	int	0 or 1	Flag for flowline
iFlag_global	int	0 or 1	Flag for global simulation
iFlag_multiple_outlet	int	0 or 1	Flag for multiple basin simulation
iFlag_use_mesh_dem	int	0 or 1	Flag to use DEM within the mesh
iFlag_elevation_profile	int	0 or 1	Flag to turn on elevation profile
iFlag_rotation	int	0 or 1	Flag for mesh generation using rotation

continues on next page

Table 2 – continued from previous page

Keyword	Data type	Value	Description
iFlag_stream_burning_topo	int	0 or 1	Flag to turn on stream burning topology
iFlag_save_elevation	int	0 or 1	Flag to save elevation
iCase_index	int		ID of case
iMesh_type	int	1 to 5	Mesh type
dMissing_value_dem	float		The missing value in the DEM
dBreach_threshold	float		The threshold parameter for the hybrid breaching filling algorithm
dAccumulation_threshold	float		The accumulation parameter to define stream cell
dLongitude_left	float	0 or 1	The domain left boundary
dLongitude_right	float		The domain right boundary
dLatitude_bot	float		The domain bottom boundary
dLatitude_top	float		The domain top boundary
dResolution_degree	float		Mesh resolution in degree
dResolution_meter	float		Mesh resolution in meter
sDate	string		The date of the simulation
sMesh_type	string	hexagon	The mesh type
sFilename_hexwatershed	string		The filename of the binary
sFilename_spatial_reference	string		The spatial reference of the river network
sFilename_dem	string		The filename of the DEM
sFilename_mesh_netcdf	float		The filename of the MPAS netcdf file
sFilename_basins	string		The full path of the basin configuration file

Basin configuration

The basin configuration file contains one or more block of JSON basin object. Each block contains the configuration to a unique basin. Different basin may have different parameters.

Table 3: Basin configuration JSON file

Keyword	Data type	Value	Description
dLatitude_outlet_degree	float		The outlet latitude
dLongitude_outlet_degree	float	0 or 1	The outlet longitude
dAccumulation_threshold	float		The accumulation parameter to define the stream cell
dThreshold_small_river	float		The threshold parameter to remove small river
iFlag_dam	int	0	Reserved for dam burning
iFlag_disconnected	int	0	Reserved for disconnected flowline
lBasinID	long		The basin ID
sFilename_dam	string		Reserved for dam burning
sFilename_flowline_filter	string		The filename of the stream vector
sFilename_flowline_raw	string		The filename of the raw stream vector
sFilename_flowline_topo	string		Reserved for dam burning

1.3.5 Model simulation

The easiest way to setup a simulation is to use an existing template. You can also generate an emtpy template using the provided APIs.

Then you can edit the template by replacing with the actual input filenames and paths.

Last, you can run the model through the Python APIs.

1.3.6 Simulation results

After the simulation is finished, you should obtain a list of fils within the output directory. Depending on the configuration, not all files will be outputed.

- depression filled DEM
- flow direction
- flow accumulation
- stream segment
- stream order
- subbasin boundary
- watershed boundary

These files are saved using the GeoJSON file format.

Table 4: Domain-scale output option

Vector type	Variable	Global	Multiple outlets	Single outlet
Point	Dam	No	No	No
Polyline	Flow direction	Yes	Yes	No
Polyline	Stream segment	Yes	Yes	No
Polygon	Elevation	Yes	Yes	No
Polygon	Slope	Yes	Yes	No
Polygon	Drainage area	Yes	Yes	No
Polygon	Travel distance	Yes	Yes	No

Table 5: Watershed-scale output option

Vector type	Variable	Global	Multiple outlets	Single outlet
Point	Dam	No	No	No
Polyline	Flow direction	No	Yes	Yes
Polyline	Stream segment	No	Yes	Yes
Polygon	Elevation	Yes	Yes	Yes
Polygon	Slope	Yes	Yes	Yes
Polygon	Drainage area	Yes	Yes	Yes
Polygon	Travel distance	No	Yes	Yes

You can use any GIS tools (ArcGIS, ENVI, and QGIS, etc.) to visualize the results.

1.3.7 Tutorial

A full tutorial is provide at https://github.com/changliao1025/hexwatershed_tutorial

1.4 Algorithm

1.4.1 Overview

HexWatershed includes major classicial watershed delineation algorithms with modifications.

These algorithms are implemented within both the Python frontend and the C++ backend.

1.4.2 Algorithms

Distance

Distance is calculated using the great-circle distance, orthodromic distance, or spherical distance is the distance along a great circle.

Area

A closed geometric figure on the surface of a sphere which is formed by the arcs of great circles.

Angle

A spherical angle is a particular dihedral angle; it is the angle between two intersecting arcs of great circles on a sphere.

Flow simplication

Flowline simplication is achieved through the PyFlowline Python package (https://anaconda.org/conda-forge/pyflowline)

Mesh generation

Mesh generation is achieved through the PyFlowline Python package (https://anaconda.org/conda-forge/pyflowline)

Topological relationship reconstruction

Topological relationship reconstruction is achieved through the PyFlowline Python package (https://anaconda.org/conda-forge/pyflowline)

DEM resampling

HexWatershed provides two resampling methods:

Nearest resampling

In the nearest resampling, the model defines the cell elevation in the following steps:

- obtain cell center location in **longitude** and **latitude**
- convert location into DEM row and column indices
- obtain DEM pixel value and assign as the cell elevation

Zonal mean resampling

In the zonal mean resampling, the model defines the cell elevation in the following steps:

- obtain all the cell vertex locations in **longitude** and **latitude**
- define a polygon using the vertex locations
- extract the DEM using the polygon
- calculate the mean extracted DEM and assign as the cell elevation

Stream burning

A topological relationship-based stream burning algorithm is implemented in HexWatershed v3.0.

Depression filling

In general, the depression filling algorithm is similar to that in the HexWatershed v1.0.

The major difference is related to the stream burning algorithm.

Flow direction

The flow direction is defined based on elevation differences and distances.

Currently, only the deepest slope is used to define the single flow direction.

Multi-flow directions will be supported in future versions.

Flow accumulation

The actual drainage area is used instead of flow accumulation cell number. This is because each cell may have different cell area.

Stream grid

The flow accumulation threshold is used to define the stream grid.

River confluence

If a stream grid has more than one upstream, this stream grid is defined as a river confluence.

Stream segment

Each individual stream line between headwater-confluence is defined as a stream segment

Subbasin

All the cells that contributes to a stream segment are used to define a subbasin.

Watershed

All the cells that contributes to a river outlet are used to define a watershed.

1.5 History

- 2017-05-12: Design
- 2020-04-12: Publish

1.6 Support

Limited support is provided through Issue:

and Slack:

1.7 Contribution

HexWatershed was developed and maintained by

- Chang Liao (Pacific Northwest National Laboratory)

1.8 API Reference

1.8.1 C++ API Docs

Overview

C++ class

JSON

class **cell** : public jsonmodel::JSONBase

Public Members

std::vector<long> **aNeighbor**
neighbor ID

std::vector<long> **aNeighbor_land**
land neighbor ID

std::vector<long> **aNeighbor_ocean**
ocean neighbor ID

std::vector<float> **aNeighbor_distance**
neighbor distance

float **dElevation_mean**
: average elevation

float **dElevation_profile0**
: elevation profile

float **dElevation_raw**
: original elevation

float **dLatitude_center_degree**
: latitude

float **dLongitude_center_degree**
: longitude

float **dArea**
: cell area

float **dAccumulation**
: flow accumulation

float **dSlope_between**
: slope between this cell and downslope cell

float **dSlope_within**
: slope based on high resolution DEM

float **dSlope_profile**
: slope based on elevation profile between downslope cell

float **dLength_flowline**
: flowline length

float **dLength**
: effective cell length

float **dDistance_to_downslope**
: distance to downsloe

float **dDistance_to_subbasin_outlet**
: distance to subbasin outlet

float **dDistance_to_watershed_outlet**
: distance to watershed outlet

int **nVertex**

number of vertex

long **lCellID**

global cell ID

long **lCellID_downstream_burned**

pre-descibed global downstream cell ID

long **lCellID_downslope**

global downslope cell ID

class **vertex**

Public Functions

bool **operator==**(const *vertex* &cVertex)

overload the equal function

Parameters
cVertex –

Returns
true

Returns
false

float **calculate_slope**(*vertex* pt)

calculate the slope between two vertices

Parameters
pVertex_in –

Returns
float

Domain

class **domain**

Public Functions

domain(std::string sFilename_configuration_in)

Parameters
sFilename_configuration_in – user provided model configuration file please refer to the user guide for I/O instruction

int **domain_read**()
read data from the model configuration file

Returns

int **domain_read_configuration_file**()
read the user provided configuration file

Returns

int **domain_read_input_data**()
read input data

Returns
int

int **domain_retrieve_user_input**()
extract the dictionary from user provided configuration file

Returns

int **domain_run**()
run the model

Returns

int **domain_cleanup**()
clean up the model status

Returns

Compset

class **compset**

Public Functions

int **compset_initialize_model**()
initialize the model

Returns

int **compset_save_model**()
save all the model outputs

Returns

int **compset_priority_flood_depression_filling**()

DEM depression filling

Returns

int **compset_stream_burning_with_topology**(long lCellID_center)

Parameters
lCellIndex_center –

Returns
int

int **compset_stream_burning_without_topology**(long lCellID_center)

Parameters
lCellIndex_center –

Returns
int

int **compset_breaching_stream_elevation**(long lCellID_center)

Parameters
lCellID_active –

Returns
int

int **compset_calculate_flow_direction**()

calculate the flow direction based on elevation, this step "should" only be run after the depression filling

Returns

int **compset_calculate_flow_accumulation**()

calculate the flow accumulation based on flow direction

Returns

int **compset_define_stream_grid**()

define the stream network using flow accumulation value

Returns

int **compset_define_watershed_boundary**()

define the watershed boundary using outlet

Returns

int **compset_define_stream_confluence**()

define the stream confluence point because we need to topology info, the vCell_active will be used

Returns

int **compset_define_stream_segment**()

define the stream segment, must use vCell_active

Returns

int **compset_define_subbasin**()

define subbasin boundary, it requires cell topology, so the vCell_active is used

Returns

int **compset_calculate_watershed_characteristics**()

Returns
int

int **compset_save_watershed_characteristics**()

save the watershed characteristics in the output

Returns

std::vector<*hexagon*> **compset_obtain_boundary**(std::vector<*hexagon*> vCell_in)

retrieve the boundary of the hexagon grid boundary it is not ordered

Parameters
vCell_in – :the hexagon grid

Returns

int **find_land_ocean_interface_neighbors**(long lCellID_in)

Parameters
lCellID_in –

Returns
int

int **compset_update_cell_elevation**()

Returns
int

int **compset_update_vertex_elevation**()

Returns
int

int **compset_check_digital_elevation_model_depression**(std::vector<*hexagon*> vCell_in)

private functions. check whether there is local depression in the dem or not. in fact, a more rigorous method should pass in dem instead of the hexagon vector but because we will not change any member variable here, it should be safe to pass in the vector

Parameters
vCell_in –

Returns

std::array<long, 3> **compset_find_lowest_cell_in_priority_queue**(std::vector<*hexagon*> vCell_in)

find the hexagon with the lowest elevation

Parameters
vCell_in – :the hexagon grid

Returns

General

class **edge**

Public Functions

int **calculate_length**()

calculate the arc length of an edge on a sphere

Returns
int

int **check_point_overlap**(vertex pt)

check whether a vertex is one the edge or not

Parameters
pVertex_in –

Returns
int

int **check_overlap**(vertex pt_start, vertex pt_end)

check whether an edge overlap with another edge, this algorithm has error

Parameters

- **pVertex_start** –
- **pVertex_end** –

Returns
int

int **check_shared**(*edge* ed)

check whether two edge are the same ignoring the direction

Parameters
pEdge_in –

Returns
int

class **flowline**

Public Functions

int **share_vertex**(*flowline* pFlowline_in)

check whether two flowlines share a starting or ending vertex

Parameters
pFlowline_in –

Returns
int

int **share_vertex**(*flowline* pFlowline_in, vertex pVertex_in)

check two flowlines share the specified vertex

Parameters

- **pFlowline_in** –
- **pVertex_shared** –

Returns
int

class **hexagon**

Public Functions

int **calculate_average_edge_length**()

calculate the mean edge length

Returns
int

int **calculate_effective_resolution**()

calculate the effective resolution using area

Returns
int

int **update_location**()

update the x y z location

Returns
int

Public Members

long **lCellID**

Brief description. this is the mesh id from the json, it might be the same with Global ID,

Detailed description starts here.

class **segment**

Public Functions

int **calculate_stream_segment_characteristics**()

Returns
int

int **calculate_stream_segment_length**()

calculate stream segment length

Returns

int

class **subbasin**

class **watershed**

Public Functions

int **watershed_build_stream_topology**()

build the stream topology based on stream segment information

Returns

int **watershed_define_stream_order**()

build the stream order based on stream topology

Returns

int **calculate_watershed_characteristics**()

calculate the watershed characteristics

Returns

int **calculate_watershed_drainage_area**()

calculate the watershed drainage total area we can either sum up hexagon or sum up subbasin

Returns

int **calculate_watershed_total_stream_length**()

calculate the total stream length

Returns

int **calculate_watershed_longest_stream_length**()

calculate the longest stream length

Returns

int **calculate_watershed_drainage_density**()

calculate the watershed area to stream length ratio

Returns

int **calculate_watershed_average_slope**()

calculate the mean slope of the watershed we can use either subbasin or each cell

Returns

int **calculate_topographic_wetness_index**()

calculate the TWI index using method from //https://en.wikipedia.org/wiki/Topographic_wetness_index // {\displaystyle \ln {a \over \tan b}}

Returns

int **save_watershed_characteristics**()

save the watershed characteristics in the output

Returns

Data

class **data**

Public Static Functions

static float ***read_binary**(std::string sFilename_in)

read_eco3d binary file (float type)

Parameters

sFilename_in –

Returns

float*

static std::vector<float> **read_binary_vector**(std::string sFilename_in)

read_eco3d binary and save to a vector

Parameters

sFilename_in –

Returns

vector<float>

static int **write_binary_vector**(std::string sFilename_in, vector<float> vData_in)

write vector to float binary file

Parameters

- **sFilename_out** –
- **vData_in** –

Returns

int

C++ function

CHAPTER

TWO

ADDENDUM

2.1 Glossary

DEM
: Digital elevation model.

Spatial discretization
: Subdivision of the computational domain in a finite number of control volumes or elements (i.e., the generation of the numerical grid).

Stream order
: The stream order or waterbody order is a positive whole number used in geomorphology and hydrology to indicate the level of branching in a river system.

Watershed
: A drainage basin is any area of land where precipitation collects and drains off into a common outlet, such as into a river.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

D

H

J

S

W