Chang Liao
Pacific Northwest National Laboratory

WILEY-
INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Title, etc
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To my family

CONTENTS IN BRIEF

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

ECO3D was born during my PhD program working with Prof. Zhuang in the Ecosystems and Biogeochemical Dynamics Laboratory (EBDL) group in Department of Earth, Atmospheric, and Planetary Sciences (EAPS), Purdue University, Indiana, USA.

During my work in land surface modeling (LSM) and groundwater-surface water hydrology modeling, I felt there are huge gaps (spatial-temporal resolutions and model assumptions, etc.) between them. Among them, lateral flow process is generally omitted in LSM modeling due to coarse spatial resolution whereas it is unanimously considered in hydrology modeling. Therefore, I decided to close the gaps through building a LSM model that explicitly considers the lateral flow including both water and carbon fluxes. Then the ECO3D was developed.

CHANG LIAO

Richland, Washington
September, 2017

ACKNOWLEDGMENTS

This study is supported through projects funded to Qianlai Zhuang by the NASA Land Use and Land Cover Change program (NASA-NNX09AI26G), Department of Energy (DE-FG02-08ER64599), the NSF Division of Information and Intelligent Systems (NSF-1028291), and a project from the United States Geological Survey focusing on quantification of carbon and methane dynamics in Alaska. This research was supported in part through computational resources provided by Information Technology at Purdue, West Lafayette, Indiana.

GLOSSARY

reach	A stream segment.
permafrost	Frozen soil.
HRU	A basic computational unit assumed to be homogeneous in hydrologic response to land cover change.

ACRONYMS

CU	Column Unit
DEM	Digital Elevation Model
HRU	Hydrologic Response Unit
MODIS	Moderate-resolution imaging spectroradiometer
PAR	photosynthetically active radiation
PRMS	Precipitation-Runoff Modeling System
SW	shortwave radiation

INTRODUCTION

Earth is NOT flat, it is a three-dimensional world. As a result, nearly all natural processes and human activities are influenced by its three-dimensional surroundings.

However, due to limitations in science and technology, we usually have to project our world onto a one or two-dimensional domain in real life practices, such as Google Map.

In Earth Science community, specifically in land surface modeling (LSM) group, most model frameworks only consider vertical flux (i.e., carbon and water fluxes) exchanges and assume that there is no horizontal interactions. There are several reasons supporting this assumption. First, land surface modeling has only developed for few decades and most current work are based on first generation model framework, which is rather simplified due to the computational capabilities in the 1990s. Second, nearly all land surface models are using coarse spatial resolutions (e.g., $1.0^\circ \times 1.0^\circ$) and therefore horizontal flux terms are small compared with vertical flux terms.

Pushed by the increasing demand for high spatial-temporal resolution products, more and more modeling work have started to re-evaluate this assumption and consider using fully three-dimensional approaches, especially when surface topography plays an important role.

Moreover, in hydrology community, which is one of the oldest science disciplines, lateral interactions are always considered because they essentially form the stream flow.

To date, fully coupled frameworks which consider both land surface processes and hydrological processes are rare. If we want to estimate the water and carbon cycle at high spatial-temporal resolutions, a fully three-dimensional approach is irreplaceable. And that is why I conducted this study.

The objective of this study is to establish a flexible framework to estimate water and carbon cycle at high spatial-temporal resolutions.

The objective of this report is to provide a theoretical and technical documentation for the three-dimension ecosystem model (ECO3D) for both researchers and developers.

The structure of this report is organized based on the actual structure of the ECO3D model. An easy way to understand the structure is to take a look at the dependence map or the source code makefile. Throughout the report, lots of source codes are directly embedded within text for reference.

PART I

SUPPORTING LIBRARY

CHAPTER 1

MODEL STRUCTURE

1.1 Overview

The ECO3D model is developed based on land surface model and hydrology model using a fully coupled three-dimensional approach. Therefore, great care has been taken to make sure the model is easy-to-use for both users and developers.

1.2 Development Environment

In order to improve code readability, the modular programming (https://en.wikipedia.org/wiki/Modular_programming) and object-oriented programming (https://en.wikipedia.org/wiki/Object-oriented_programming) techniques are used in designing the model structure.

Besides, I decided to use a programming language that is fast. Because a fully three-dimensional model implies significant computational demand.

For high performance computing, I decided to use OpenMP and MPI whenever possible, which is widely used in Earth system modeling.

In the end, the C++11 programming language is used as the main development language. It is object-oriented through various data types including class, which also supports modular programming recently. C++11 is fully supported in multiple platforms including Unix and Linux.

OpenMPI can be enabled within C++11 without significant amount of effect. MPI can also be used if necessary. However, due to the interactions between neighboring Column Unit (CU), special attention must be taken when implementing MPI.

1.3 Structure of ECO3D

Based on my choice of development environment, the ECO3D model is broken into several components.

First, I will list a few base/global classes, which supports various functionalities for various model components.

Then, I will introduce the core concept, i.e., Column Unit (Chapter 11), in the model and describe how this concept is implemented throughout the model.

Afterwards, the report will explain how each class is implemented with depth.

As last, I will provide a complete demonstration of the model simulation.

A list of optional tools are also introduced to advance the model preparation and application.

CHAPTER 2

DIMENSIONAL VARIABLE

2.1 Overview

The dimensional component stores all the important dimension variables (e.g., numbers of column and row of the spatial extent). All the dimension variables are constant throughout the simulation at current version.

2.2 Dimensions

Among all the dimension variables, some of them are defined based on model structure. Therefore, they are constants regardless of the study area.

Table 2.1 Dimension variables used in the ECO3D model.

Name	Value	Description
$nmonth$	12	Number of month within a year
$nlandcover\_type$	16	Number of land cover type
$nsoil\_type$	12	Number of soil type
$nsoil\_layer$	6	Number of soil layer
$nvegetation\_type$	12	Number of vegetation type
$nhru\_type$	5	Number of HRU type
$column\_type$	5	Number of Column Unit type

Some other dimension variables are defined based on the study area, such as the numbers of column and row of the study area spatial extent.

Table 2.2 Dimension variables based on study area.

Name	Value	Description
$ncolumn$	388	Number of column in the spatial extent
$nrow$	310	Number of row in the spatial extent
$nsegment$	37	Number of stream segment

Because the dimensional variables based on study area can also be treated as user inputs, they can be defined dynamically in future version.

CHAPTER 3

GLOBAL VARIABLE

3.1 Overview

A number of variables other than dimensions are also used by several components, therefore, an independent class was defined to store these global variables.

Global variables can be generally classified into several categories based on attributes and usages.

3.2 Mathematic Constants

Table 3.1 Math constants.

Name	Value	Unit	Description
Pi	3.1415926	$unitless$	The famous pi
$near\_zero$	1.0^{-6}	$unitless$	Used as a threshold
$missing\_value$	-9999.0	$unitless$	The default value for missing value for output

3.3 Time

Table 3.2 Time.

Name	Value	Unit	Description
day_2_second	3600×24	$unitless$	Second in a day
$day\_2\_minutes$	60×24	$unitless$	Minute in a day
$hour\_2\_second$	3600	$unitless$	Second in an hour
$dTimestep\_ecosystem$	1.0	day	The default time step for all processes

3.4 Length

Table 3.3 Length.

Name	Value	Unit	Description
$meter\_2\_millemeter$	1.0^3	$unitless$	
$meter\_2\_foot$	3.28084	$unitless$	
$millimeter\_2\_meter$	1.0^{-3}	$unitless$	
$inch\_2\_meter$	2.54×1.0^{-2}	$unitless$	
$inch\_2\_centimeter$	2.54	$unitless$	

3.5 Area

Table 3.4 Area.

Name	Value	Unit	Description
$square\_meter\_2\_square\_centimeter$	1.0^4	$unitless$	
hpa_2_pa	1.0^2	$unitless$	

Table 3.5 Mass.

Name	Value	Unit	Description
$gram\_2\_kilogram$	1.0^{-3}	$unitless$	
$milligram\_2\_kilogram$	1.0^{-6}	$unitless$	

3.6 Mass

3.7 Temperature

Table 3.6 Temperature.

Name	Value	Unit	Description
$kelvin\_2\_celsius$	-273.15	$unitless$	
$celsius\_2\_kelvin$	273.15	$unitless$	

3.8 Unit Conversion

Table 3.7 Unit conversion.

Name	Value	Unit	Description
$cubic\_meter\_2\_cubic\_centimeter$	1.0^{6}	$unitless$	
$cubic\_meter\_2\_cubic\_liter$	1.0^{3}	$unitless$	
$cubic\_centimeter\_2\_cubic\_meter$	1.0^{-6}	$unitless$	
$kilogram\_per\_kilogram\_2\_milligram\_per\_gram$	1.0^{3}	$unitless$	
$milligram\_per\_gram\_2\_kilogram\_per\_kilogram$	1.0^{-3}	$unitless$	
$kilogram\_per\_cubic\_meter\_2\_gram\_per\_cubic\_centimeter$	$1.0^{3}/1.0^{6}$	$unitless$	
$kilogram\_per\_cubic\_meter\_2\_gram\_per\_liter$	1.0	$unitless$	
$kilogram\_per\_cubic\_meter\_2\_milligram\_per\_liter$	1.0^{3}	$unitless$	
$milligram\_per\_liter\_2\_kilogram\_per\_cubic\_meter$	1.0^{-3}	$unitless$	
$kilogram\_per\_square\_meter\_2\_gram\_per\_square_{m}eter$	1.0^{3}	$unitless$	
$gram\_per\_square\_meter\_2\_kilogram\_per\_square_{m}eter$	1.0^{-3}	$unitless$	
$calorie\_2\_joule$	4.1858	$unitless$	
$joule\_2\_calorie$	$1/4.1858$	$unitless$	
$joule\_2\_megajoule$	1.0^{-6}	$unitless$	
$joule\_2\_langley$	$1.0/41840.0$	$unitless$	
$langley\_2\_joule$	41840.0	$unitless$	

3.9 Physical Constants

Table 3.8 Physical constants.

Name	Value	Unit	Description
$dStefan\_boltzmann$	5.670373×1.0^{-8}	$w \times m^{-2}K^{-4}$	
$dManning\_roughness$	0.02	$unitless$	
$radian$	0.0172	$unitless?$	
$dEccentricy$	0.0167	$unitless?$	
$dSolar\_constant$	1368.0	??	
$dFrozen\_temperature$	273.15	K	
$dDensity\_water$	1.0^3	$kg \times m^{-3}$	
$dLatent\_heat\_water$	333.55×1.0^3	$j \times kg^{-1} \times K^{-1}$	
$dSpecific\_heat\_water$	4.179×1.0^3	$j \times kg^{-1} \times K^{-1}$	
$dSpecific\_heat\_ice$	2.03×1.0^3	$j \times kg^{-1} \times K^{-1}$	

3.10 GIS

Table 3.9 GIS.

Name	Value	Unit	Description
$dResolution$	500.0	m	
$dArea$	500.0×500.0	m^2	

3.11 Module

Table 3.10 Module.

Name	Value	Unit	Description
$dTemperature\_all\_rain$	273.15	K	
$dTemperature\_all\_snow$	273.15	K	
$dFraction\_sublimation$	0.5	$unitless$	

3.12 System

Table 3.11 System.

Name	Value	Unit	Description
$slash$	$'/'$	$unitless$	Used for cross platform compilation

CHAPTER 4

CONVERSION

4.1 Overview

The conversion class mainly deals with conversions between different systems (e.g., unit system) using various functions. While most conversions can be done directly using provided global constants, some require a simple function to achieve the goal.

4.2 Member Function

4.2.1 Unit System

For most of the unit system conversions, they are done directly using the conversion coefficients from the global variables (Chapter 3).

For those cannot be converted directly using the global variables, a number of functions are used.

1. Kelvin to Fahrenheit

```
double convert_from_kelvin_to_fahrenheit(double
    dTemperature_kelvin);
```

2. Fahrenheit to Kelvin

```
double convert_from_fahrenheit_to_kelvin(double
    dTemperature_fahrenheit);
```

Details of these functions are presented within the source code.

4.2.2 String Operation

A number of functions are used to convert between various data types and strings.

1. Integer to string

```
std::string convert_integer_to_string(int iNumber);
```

2. Integer to string with width

```
std::string convert_integer_to_string(int iNumber,
  int iWidth);
```

3. Double to string

```
std::string convert_double_to_string(int iPrecision,
  int iWidth,
  double dNumber);
```

4. Split string

```
std::vector<std::string> split_string_by_space(std::string
    sString_in);
```

5. Split string by a delimiter

```
std::vector<std::string> split_string_by_delimiter(std::string
     sString_in, char cDelimiter);
```

Details of these functions are presented within the source code.

CHAPTER 5

DATA I/O

5.1 Overview

The data class mainly deals with data/file related operations. In the ECO3D model, I try to use standard file format (i.e., binary) whenever possible.

5.2 Data Format

The binary format is used in the ECO3D model whenever possible in current version. The format is also widely used in ENVI (https://www.harris.com/solution/envi) and ArcGIS (https://www.arcgis.com) software program. Detailed documentation of this format is discussed here (http://harrisgeospatial.com/docs/ENVIHeaderFiles.html) and there (http://pro.arcgis.com/en/pro-app/tool-reference/conversion/raster-to-float.htm).

Even though most input data used in the ECO3D model are in single-precision float format, we currently use the double-precision throughout the simulation to guarantee the precision without sacrificing much performance. In the future, we plan to tailor the program to improve the performance through better memory management.

Besides, when we output the simulation variables, all double-precision float data were converted back to single precision.

Last, as data I/O is an independent class, all the functions are defined as static members.

5.3 GIS compatibility

To ease the effort for data management and visualization, data flow within the model development is designed in a way that no extra processes are needed to visualize the results. Specifically, all output variables are in GIS visualization ready format and all inactive regions are set to missing value by default.

5.4 Member Function

A number of member functions are defined:

1. Read binary file

```
static float * read_binary(std::string sFilename);
```

2. Write binary file

```
static int write_binary_vector(std::string sFilename,
    vector <double> vData);
```

Details of these functions are presented within the source code.

In the future, I will add support for the PETSc library (https://www.mcs.anl.gov/petsc/) for high performance data I/O and mathematical operation (Chapter 7). Also, additional library may be added to support other popular formats such as HDF (https://www.hdfgroup.org/) and NetCDF (https://www.unidata.ucar.edu/software/netcdf/).

Also, I am considering using cloud database to data management in future version.

CHAPTER 6

DATE/TIME

6.1 Overview

The date class mainly deals with all date/time related operations.

An existing open source C++ library was used directly to speed up the development. This library can be acquired through here (http://technical.silverglass.org).

6.2 Member Function

A number of member functions are defined:

1. Date to julian day

```
static long YmdToJd( const int iYear, const int iMonth, const
    int iDay);
```

2. Is it a leap year

```
static int IsLeapYear( const int iYear );
```

3. Day of year

```
static int DayOfMonth (int year,int month);
```

For more details of these functions, please refer to the source code.

CHAPTER 7

MATHEMATICS

7.1 Overview

The mathematics class mainly deals with various mathematical operations.

Because only a number of simple mathematical calculations are required currently (most processes are simulated using explicit finite difference method), I did not use third-party mathematics library so far. In the future, I will add supports for third party libraries such as PESTc (https://www.mcs.anl.gov/petsc/).

7.2 Member Function

A number of member functions are defined.

1. Mean of vector

```
double calculate_mean(std::vector<double> data);
```

2. Mean of vector excluding missing value

```
double calculate_mean(std::vector<double> data,
                      double missing_value);
```

3. Mean of vector in a range

```
double calculate_mean(std::vector<double> data,
                      double max_value,
                      double min_value);
```

4. Matrix solver

```
void tridia(int n,
            vector<double> a,vector<double> b,
            vector<double> c,vector<double> r,
            vector<double> u);
```

Details of these functions can be found in the source code.

CHAPTER 8

OPERATION SYSTEM

8.1 Overview

The system class mainly deals with various operations associated with operation systems.

8.2 Cross Platform Compatibility

Because the ECO3D model was written using C++11, it could be compiled on most operation systems, which is implemented through preprocessor directives (http://www.cprogramming.com/reference/preprocessor/).

```
//cros-platform header
#ifdef _WIN32
//define something for Windows (32-bit and 64-bit, this part is
    common)
#include <direct.h>
#include <windows.h>
#ifdef _WIN64
//define something for Windows (64-bit only)
#else
//define something for Windows (32-bit only)
#endif
#elif __APPLE__
#include <sys/stat.h>
#include <TargetConditionals.h>
#if TARGET_IPHONE_SIMULATOR
// iOS Simulator
#elif TARGET_OS_IPHONE
// iOS device
#elif TARGET_OS_MAC
// Other kinds of Mac OS
#else
#  error "Unknown Apple platform"
#endif
#elif __linux__
// linux
#include <sys/stat.h>
#include <sys/types.h>
#elif __unix__ // all unices not caught above
// Unix
#elif defined(_POSIX_VERSION)
// POSIX
#else
#   error "Unknown compiler"
#endif
```

Bear in mind that even though the ECO3D model can be compiled under various platforms, I have only tested Windows, Linux and macOS for current version.

8.3 Member Function

A number of member functions are defined:

1. Test whether a file exist or not

```
int file_test(std::string sFilename);
```

2. Create a directory recursively

```
int make_directory(std::string sDirectory);
```

3. Test whether a path exist or not

```
int path_test(std::string sPath);
```

4. System call to run a command

```
int run_command(std::string sCommand);
```

5. Get the size of a file

```
long get_file_size(std::string sFilename);
```

Details of these functions are presented within the source code. All functions are implemented with cross platform capability.

PART II

MATHEMATICAL THEORY

CHAPTER 9

WATER CYCLE

9.1 Overview

There are several processes that are critical in our water cycle model.

In a word, the current model has used a number of empirical methods to represent some really complicated hydraulic processes.

On land surface, the overland surface runoff is estimated using a continuum equation. However, a kinetic wave equation may be more accuratee.

In subsurface, the soil zone interflow is estimated based on based from PRMS. A variably saturated flow approach could produce better results.

In stream routing, the muskingum stream routing method is used. In the future, the kinetic wave method may be provided as another option.

9.2 Overland Surface Runoff

Currently, the surface runoff algorithm is from the PRMS model. In this module, the continuum equation is developed based on various fluxes.

$$\begin{aligned}\frac{dStorage_{water}}{dt} = &\sum Inflow_{upslope} \\ &+ Inflow_{snowmelt} \\ &+ Inflow_{throughfall} \\ &- Outflow_{downslope} \\ &- Outflow_{evaportranspiration} \\ &- Outflow_{infiltration}\end{aligned} \tag{9.1}$$

Among them, several flux terms are spatial dependent due to the topography effects. Therefore, they are calculated outside the normal processes, i.e. using the cascade module.

Due to the coupling of water flux and dissolved organic carbon, the continuum equation of carbon and dissolved organic carbon are modelled using the similar framework.

For carbon, the continuum is as follow:

$$\begin{aligned}\frac{dStorage_{carbon}}{dt} = &\sum Inflow_{upslope} \\ &+ Inflow_{snowmelt} \\ &+ Inflow_{throughfall} \\ &- Outflow_{downslope} \\ &- Outflow_{evaportranspiration} \\ &- Outflow_{infiltration}\end{aligned} \tag{9.2}$$

For dissolved organic carbon, the continuum equation will be a little more complicated because of the concentration and flow relationship:

$$\begin{aligned}\frac{dStorage_{doc}}{dt} = &\sum Inflow_{upslope} \\ &+ Inflow_{snowmelt} \\ &+ Inflow_{throughfall} \\ &- Outflow_{downslope} \\ &- Outflow_{evaportranspiration} \\ &- Outflow_{infiltration}\end{aligned} \tag{9.3}$$

All the terms are mass-based instead of concentration-based.

9.3 Stream Routing

Details of the stream algorithm can be found within the PRMS mannual.

CHAPTER 10

CARBON CYCLE

PART III

MODEL COMPONENT

CHAPTER 11

COLUMN UNIT

11.1 Overview

The most important concept within the ECO3D model is the Column Unit (CU), which is similar to the concept of grid cell in other Earth system models. However, in a three-dimensional model structure, Column Unit emphasizes the interactions between adjacent units based on its characterictics.

11.2 Definition

By definition, the Column Unit is the cube that occupies the whole grid cell from atmosphere to bedrock.

Each Column Unit is characterized by its vertical profile based on CU classification. For example, the most common vertical profile includes atmosphere, vegetation canopy, litter and soil. In other scenarios, a Column Unit vertical profile may include lake or stream. Because different types of vertical profile are associated with significant different types of processes, we define several Column Unit types to consider these dynamics.

In current version, five Column Unit types are defined:

Table 11.1 Type of Column Unit.

Type	Description
Glacier	A CU containing glacier
Lake	A CU containing lake
Land	A CU containing land
Stream	A CU containing stream reach
Swale	A CU containing local depression

In future development, other CU types may be added.

Because atmosphere component is shared by all CU types, I separated it from the Column Unit object and consider it as an independent component in model structure.

CHAPTER 12

PROGRAM ENTRANCE

12.1 Overview

The main program is the entrance of the ECO3D model. It deals with important operations related to user interface and process controls.

12.2 Member Function

12.2.1 Debugging Environment

The main program defines the IEEE float precision related standard for model debugging process. [1] For example, if any variable becomes invalid during the runtime, the program will stop and exit with warnings.

```
//=======================================================
//control the float precision.
//=======================================================
feenableexcept(FE_INVALID | FE_OVERFLOW);
```

[1]This operation is also defined with cross platform compatibility.

12.2.2 User Input

```
//=====================================================
//verify the user inputs
//=====================================================
std::cout<<"Start to run ecosystem model!"<<std::endl;
int program_status = 1;
std::string sConfiguration_file = "";
//=====================================================
//examine the number of arguments provided
//=====================================================
if(argc == 2) //at least 2 arguments are needed
  {
    std::cout<<"The following arguments are provided:"<<std::
        endl;
    std::cout<<argv[1]<<std::endl; //print out all the arguments
    sConfiguration_file = argv[1];
  }
else
  {
    std::cout<<"No arguments are provided!"<<std::endl;
    std::cout<<"Please input the configuration file:"<<std::endl
        ;
    std::cin>>sConfiguration_file;
  }
```

12.2.3 Simulation Control

```
//=====================================================
//setup the model
//=====================================================
if( cEcosystem.setup() != 1)
  {
    exit(0);
  }
//=====================================================
//read input data
//=====================================================
if( cEcosystem.read() != 1)
  {
    exit(0);
  }
//=====================================================
//initialize other variables
//=====================================================

if (cEcosystem.initialize() != 1)
  {
    exit(0);
  }
//=====================================================
```

```
//run the simulation
//=======================================================
if( cEcosystem.run_and_save() != 1)
  {
    exit(0);
  }
//=======================================================
//release memory and dummy files
//=======================================================
if( cEcosystem.cleanup() != 1)
  {
    exit(0);
  }
```

The above script is used to control the model simulation.

12.3 The Configuration File

The configuration file contains all the model simulation required information. It has a text-based format. Each line within the file specifies an argument-value pair. The program is not sensitive to the orders of the lines. An example of the configuration file is provided in Chapter 22.

CHAPTER 13

CASCADE

13.1 Overview

The cascade module is used to prepare the cascade parameters for the ECO3D model. This module can also be run in offline mode because the cascade parameters are topography-only dependent and act like global variables (Chapter 3).

13.2 Theory

This module was developed based the Cascade Routing Tool (CRT) from USGS. In the CRT model, the D4 model is used to consider horizontal cascade.

13.3 Implementation

A number of member variable are defined.

13.3.1 Member Variable

```
        //=====================================================
        //the vector container to store the cascade parameter
        //=====================================================
```

```
        std::vector<std::vector<double> > vCascade_para; //saved
            for output
```

This 2D vector is used to store the D8 cascade parameter for all the Column Units.

13.3.2 Member Function

1. Calculate cascade parameter

```
//=======================================================
//calculate the cascade parameters
//=======================================================
std::vector<std::vector<double>>
calculate_cascade_parameter(std::vector<double> vBoundary,
                            std::vector<double> vDem);
```

This function is used to generate cascade parameters using the revised D8 method.

2. Check dem depression

```
//=======================================================
//check whether there is a local depression in the DEM data
//=======================================================
int check_dem_depression(std::vector<double> vBoundary,
                         std::vector<double> vDem);
```

This function is used to screen the local depression in DEM.

3. Retrieve surrounding elevation

```
//=======================================================
//retrieve the surrounding DEM data of one pixel
//=======================================================
std::array<double, 9> get_surrounding_elevation(long i, long j
    ,
                                                std::vector<double>
                                                    vBoundary,
                                                std::vector<double>
                                                    vElevation);
```

4. Read outlet of watershed

```
//=======================================================
//read the outlet location
//=======================================================
std::vector<vector<long> > read_outlet(std::string
    sOutlet_file);
```

5. Save cascade parameter to file

```
//=======================================================
//save the cascade parameters to files
//=======================================================
```

```
int save_cascade_parameter(std::string sWorkspace_data,
                           std::vector<std::vector<double>>
                              vCascade_para);
```

This function is used to save the cascade parameters into files under the data workspace.

Details of these functions can be found in the source code.

CHAPTER 14

ECO3D

14.1 Overview

The ecosystem class is the head quarter of the ECO3D model. It has implemented all the functions called by the main entrance program. It also deals with the interactions between Column Unit in the spatial domain.

14.2 Member Variable

The ecosystem class uses a number of member variables to control the user interactions and model simulation.

14.2.1 Flag Member Variable

A number of flags are used to control how the simulation will be run.

Table 14.1 Member variables defined in the ecosystem class for various flags.

Name	Default Value	Description
$iFlag\_debug$	0	The flag to control the simulation in debug mode
$iFlag\_run\_carbon\_cascade$	1	The flag to control carbon cascade
$iFlag\_run\_water\_cascade$	1	The flag to control water cascade
$iFlag\_configuration\_file$	0	The flag that indicates whether configuration file exists
$iFlag\_steady\_state$	1	The flag to control a steady state simulation
$iFlag\_run\_cascade$	1	The flag to control whether to run the cascade model
$iFlag\_run\_transient$	1	The flag to control a transient simulation
$iFlag\_soil\_carbon\_initial$	0	Whether the user provides initial soil carbon storage
$iFlag\_vegetation\_carbon\_initial$	0	Whether the user provides initial vegetation carbon storage
$iFlag\_litter\_carbon\_initial$	0	Whether the user provides initial litter carbon storage
$lColumn\_count$	-9999	Number of total Column Unit

14.2.2 Date Member Variable

Date member variables are used to control the simulation time period.

Table 14.2 Member variables defined in the ecosystem class for date.

Name	Default Value	Description
$iDay\_end\_save$	31	The end day to save simulation results
$iDay\_start\_save$	1	The start day to save simulation results
$iMonth\_end\_save$	12	The end month to save simulation results
$iMonth\_start\_save$	1	The start month to save simulation results
$iYear\_end\_save$	2016	The end year to save simulation results
$iYear\_start\_save$	1986	The start year to save simulation results
$iDay\_end\_ss$	31	The end day for steady state simulation
$iDay\_start\_ss$	1	The start day for steady state simulation
$iMonth\_end\_ss$	12	The end month for steady state simulation
$iMonth\_start\_ss$	1	The start month for steady state simulation
$iYear\_end\_ss$	1900	The end year for steady state simulation
$iYear\_start\_ss$	1800	The start year for steady state simulation
$iDay\_end\_tr$	31	The end day for transient simulation
$iDay\_start\_tr$	1	The start day for transient simulation
$iMonth\_end\_tr$	12	The end month for transient simulation
$iMonth\_start\_tr$	1	The start month for transient simulation
$iYear\_end\_tr$	2016	The end year for transient simulation
$iYear\_start\_tr$	1980	The start year for transient simulation

14.2.3 System Status Member Variable

System state variables are used to store the system status during simulation.

Table 14.3 Member variables defined in the ecosystem class for system status.

Name	Default Value	Description
$dCarbon\_vegetation\_watershed$	0.0	Total vegetation carbon storage in watershed
$dCarbon\_soil\_watershed$	0.0	Total soil carbon storage in watershed
$dCarbon\_litter\_watershed$	0.0	Total litter carbon storage in watershed
$dCarbon\_watershed$	0.0	Total carbon storage in watershed
$dRespiration\_heterotrophic\_watershed$	0.0	Total heterotrophic carbon respiration in watershed
$dSoil\_doc\_watershed$	0.0	Total soil dissolved organic carbon storage in watershed
$dSoil\_doc\_production\_watershed$	0.0	Total soil dissolved organic carbon production in watershed
$dSoil\_doc\_mineralization\_watershed$	0.0	Total soil dissolved organic carbon mineralization in watershed
$dLitter\_doc\_leaching\_watershed$	0.0	Total litter dissolved organic carbon leaching in watershed
$dReach\_doc\_lateral\_watershed$	0.0	Total lateral dissolved organic carbon flow in watershed
$dCarbon\_watershed\_previous$	0.0	Total previous carbon storage in watershed
$dCarbon\_vegetation\_watershed_previous$	0.0	Total previous vegetation carbon storage in watershed
$dCarbon\_soil\_watershed\_previous$	0.0	Total previous soil carbon storage in watershed
$dCarbon\_litter\_watershed\_previous$	0.0	Total previous litter carbon storage in watershed
$sFilename\_soil\_carbon\_initial$	null	The filename of the elevation aspect file
$sFilename\_vegetation\_carbon_initial$	null	The filename of the elevation aspect file
$sFilename\_litter\_carbon\_initial$	null	The filename of the elevation aspect file

14.2.4 Workspace Member Variable

A number of string variables are used to store several directories of input/output data.

Table 14.4 Member variables defined in the ecosystem class for data workspace.

Name	Default Value	Description
$sFilename\_configuration$	null	The filename of the configuration file
$sWorkspace\_data$	null	The general data workspace
$sWorkspace\_scratch$	null	The scratch workspace
$sWorkspace\_out$	null	The output workspace
$sWorkspace\_dewpoint$	null	The input workspace of dew point data
$sWorkspace\_fpar$	null	The input workspace of fpar data
$sWorkspace\_lai$	null	The input workspace of lai data
$sWorkspace\_tmax$	null	The input workspace of maximum temperature data
$sWorkspace\_tmin$	null	The input workspace of minimum temperature data
$sWorkspace\_prec$	null	The input workspace of precipitation data
$sWorkspace\_land\_cover\_type$	null	The input workspace of land use and land cover data

14.2.5 Data File Member Variable

A number of string variables are used to store global dataset filenames.

Table 14.5 Member variables defined in the ecosystem class for global datasets.

Name	Default Value	Description
$sFilename\_aspect$	null	The filename of the elevation aspect file
$sFilename\_boundary$	null	The filename of the boundary file
$sFilename\_column\_index$	null	The filename of the column unit index file
$sFilename\_dem$	null	The filename of the elevation file
$sFilename\_header$	null	The filename of header file
$sFilename\_hru\_type$	null	The filename of the hru typefile
$sFilename\_log$	null	The filename of the simulation log file
$sLog$	null	The log string
$sFilename\_latitude$	null	The filename of the latitude file
$sFilename\_soil\_type$	null	The filename of the soil type file
$sFilename\_slope$	null	The filename of the slope file
$sFilename\_stream\_reach$	null	The filename of the stream reach file
$sFilename\_stream\_segment$	null	The filename of the stream segment file
$sFilename\_stream\_order$	null	The filename of the stream order file

14.2.6 Data Member Variable

A number of vector containers are used to store global dataset 3.

Table 14.6 Member variables defined in the ecosystem class for global datasets.

Name	Default Value	Description
$vAspect$	null	The vector of elevation aspect
$vBoundary$	null	The vector of boundary definition
$vColumn\_index$	null	The vector of CU index
$vElevation$	null	The vector of elevation
$vHru\_type$	null	The vector of hru type
$vLatitude$	null	The vector of elevation
$vSlope$	null	The vector of slope
$vStream\_reach$	null	The vector of stream reach
$vStream\_segment$	null	The vector of stream segment
$vStream\_reach\_travel\_time$	null	The vector of stream reach travel time
$vStream\_reach\_width$	null	The vector of stream reach width
$vStream\_order$	null	The vector of stream order
$vSoil\_type$	null	The vector of soil type
$vLand\_cover\_type\_init$	null	The vector of land cover type
$vSoil\_carbon\_init$	null	The vector of initial soil carbon storage
$vVegetation\_carbon\_init$	null	The vector of initial vegetation carbon storage
$vLitter\_carbon\_init$	null	The vector of initial litter carbon storage
$aStream\_segment\_topology$	null	The stream topology

14.2.7 Other Member Variable

14.2.7.1 Input/Output

A map container is used to store all the configuration variables:

```
//=========================================================
//parameter map read from the configuration file
//=========================================================
std::map<std::string, std::string> mParameter;
```

Another map and vector container are used to stored output variables:

14.2.7.2 Stream

1. The stream segment topology relationship is stored using

```
    //=========================================================
    //the array to store the stream segment topology
        information
    //=========================================================
    std::array<std::array<int, 2>, nsegment>
        aStream_segment_topology;
```

2. The indices information of all reaches are stored using:

```
//=====================================================
//the indices of all stream reaches
//=====================================================
std::vector<std::vector<long>> vStream_segment_reach;
```

3. For each stream reach, a structure is defined to store various state variables:

```
//=====================================================
//the structure to store external source/sink for each
    stream reach
//=====================================================
struct stream_reach_external
{
  int iSegment;
  int iReach;
  int iYear;
  int iMonth;
  int iDay;
  double dInflow_external_daily;
  double dDoc_concentration_external_daily;
};
//=====================================================
//this vector is used to store all the stream reach
    external information
//=====================================================
std::vector<stream_reach_external> vStream_reach_external;
```

Because any stream reach may receive external source or have sink terms, this struct/vector is used to consider this factor.

14.2.7.3 Cascade

Surface and subsurface cascade parameters are stored using:

```
//=====================================================
//D8 cascade parameter
//=====================================================
std::vector<std::vector<double>> vCascade_para;
```

The cascade parameter can be read directly from the cascade module results (Chapter 13).

14.3 Setup

Because the default constructor cannot set up desirable member variables perfectly, user defined functions are used to set up the default values for both environmental variables and output variables.

1. Construct using configuration file

```
//=======================================================
//the user defined constructor
//=======================================================
ecosystem::ecosystem(std::string sFilename_configuration);
```

   The configuration file is required to construct the ECO3D model.

2. setup default variable

```
//=======================================================
//set up input
//=======================================================
int setup_default_variable();
```

   This function is used to set up the default values for the ecosystem class member variables. For example, a map type look-up table is defined for configuration file (Section 14.2.7.1).

3. setup output variable

```
//=======================================================
//set up output
//=======================================================
int setup_output_variable();
```

   This function is used to set up the output variable look-up table (Section 14.2.7.1).

14.4 Read

After the model is setup, a number of functions are used for data preparation.

1. Read configuration file

```
//=======================================================
//read configuration
//=======================================================
int ecosystem::read_configuration_file();
```

   This function is used to read the configuration file and update the look-up table (Section 14.2.7.1).

2. Read global data

```
//=======================================================
//read global data
//=======================================================
int ecosystem::read_global_data();
```

This function is used to read the global dataset (Chapter 3 and Section 14.2.6).

3. Read stream information

```
//=======================================================
//read stream segment topology
//=======================================================
int read_stream_segment_topology();
```

This function is used to read the stream segment/reach topology relationship (Section 14.2.7.2).

4. Create stream topology relationship

```
//=======================================================
//construct the stream reach topology based on input file.
//=======================================================
int ecosystem::create_stream_segment_reach_topology();
```

This function is used to define the stream reach topology vecotor based on the stream reach input file.

5. Read stream inflow

```
//=======================================================
//read stream reach inflow
//=======================================================
int read_stream_reach_inflow();
```

This function is used to read the external information for individual stream reach (Section 14.2.7.2).

6. Read cascade parameter

```
//=======================================================
//if the cascade parameters were not prepared in advance,
    we need to run the cascade module before running the
    model simulation.
//=======================================================
int ecosystem::read_cascade_parameter();
```

The ECO3D model uses the cascade parameters to simulate the surface and subsurface interactions between column units. So if cascade parameters were not present under the data directory, it must be produced in-time using the cascade module. This is done by setting the corresponding flag to 1 in the configuration file (Section 14.2.1).

14.5 Initialization

With most global dataset been read, a number of functions are used to initialize the system status.

1. Retrieve usr input

```
//=======================================================
//update member variable based on configuration map object
//=======================================================
int ecosystem::retrieve_user_input();
```

   This function is used to update the class member variables using the configuration look-up table.

2. Index output

```
//=======================================================
//get the index of the output variables from the big look-
    up table
//=======================================================
int ecosystem::index_output_variable();
```

   This function is used to define the indices for all the desired output variables in the configuration file.

3. Initialize spatial domain

```
//=======================================================
//initialize the spatial domain based on the total count
    of column
//=======================================================
int ecosystem::initialize_spatial_domain();
```

   Because the total number of column is defined by the column index data file, the spatial domain can be defined directly. In an attempt to reduce memory usage, not all matrix pixels are initialized with a Column Unit, only active pixels are defined.

4. Assign time invariant data

```
//=======================================================
//assign global variable to each column using the geology
    class object
//=======================================================
int ecosystem::assign_time_invariant_data();
```

5. Initialize ecosystem

```
//=======================================================
//initialize all other member variables of each column
    object
//=======================================================
int ecosystem::initialize_ecosystem();
```

This function is used to initialize all the member variables of each component within each Column Unit.

6. Initialize using predefined data

```
//=======================================================
//if some initial files are provided by the user, update
    corresponding
//member variables
//=======================================================
int ecosystem::initialize_ecosystem_from_initial_file();
```

If user could provide initial data for some system state variables, they can be updated after the initialization.

14.6 Run

A number of member functions are used to run the actual model simulation.

1. Run and save

```
//=======================================================
//run and simulation and save the output variable during
    simulation period.
//=======================================================
int ecosystem::run_and_save();
```

This function is used to run a model simulation and save output variables during simulation period.

2. Run SS simulation

```
//=======================================================
//run a steady state simulation if necessary
//=======================================================
int ecosystem::run_steady_state_simulation();
```

This function is used to run a steady state simulation.

3. Run TR simulation

```
//=======================================================
//run a transient simulation
//=======================================================
int ecosystem::run_transient_simulation();
```

This function is used to run a transient simulation.

4. Call sub modules

```
//=======================================================
//run all the model modules/components
//=======================================================
```

```
int ecosystem::run_modules(int iFlag_mode, int iDay_end,
    int iDay_start,  int iMonth_end, int iMonth_start, int
    iYear_end, int iYear_start);
```

This function is used to run a list of modules within the ECO3D model.

14.7 Cascade

14.7.1 Reset

Reset all cascade variables to zero. Because cascade is an accumulative algorithm (one CU could receive multiple CU cascade from upslope), the accumulative variable must be reset to zero at each time step.

1. Glacier

Table 14.7 Member variables need reset.

Variable	Description
$dSurface\_runoff\_inflow\_upslope$	The surface runoff downslope

2. Land

Table 14.8 Member variables need reset.

Variable	Description
$dSurface\_runoff\_inflow\_upslope$	The surface runoff downslope
$dDunnian\_runoff\_inflow\_upslope$	The dunnian runoff downslope
$dGroundwater\_inflow\_upslope$	The groundwater runoff downslope
$dInterflow\_inflow\_upslope$	The soil interflow runoff downslope
$dLitter\_doc\_inflow\_upslope$	The litter doc from downslope
$dSoil\_doc\_inflow\_upslope$	The soil doc from downslope

3. Stream

Table 14.9 Member variables need reset.

Variable	Description
$dDischarge\_inflow\_lateral\_daily$	The surface runoff downslope
$dDoc\_inflow\_lateral\_daily$	The dunnian runoff downslope

14.7.2 All Cascade

The cascade member functions are implemented for surface and subsurface interactions between Column Unit.

```
//=====================================================
//run the surface and subsurface interactions for all the column
     units.
//=====================================================
int ecosystem::all_cascade(long i, long j);
```

This is core function for surface and subsurface interactions for all column units.

Because each CU can have various type of CUs, different fluxes and flow terms will be cascaded depending on actual CU types. This is a general from-to combination process.

The general idea behind this function is explained here: For each central CU (A), we will loop through all of this potential 8 neighbors. For each neighbor CU (B), we will add up the lateral cascade flux from A to B. Keep in mind that both A and B can be any of the five CU types, but not all the pairs are possible or implemented. For example, land can cascade to stream, but stream cannot cascade to land.

14.7.3 From Glacier

In current setting, regardless of the To CU type, Glacier CU can only cascade surface runoff.

1. To glacier

   Only surface runoff cascade simulated from a glacier CU to another glacier CU.

2. To lake

   Not implemented currently.

3. To land

   Only surface runoff cascade

4. To stream

   Very unlikely.

5. To swale

   Not implemented.

14.7.4 From Lake

We do not have lake components yet.

14.7.5 From Land

1. To glacier

   Only surface runoff can be cascaded (In current setting, no Land CU can cascade to Glacier CU in fact.)

2. To lake

   Because there is no lake, this is currently not implemented.

3. To land

   A number of flux terms will be cascaded.

Table 14.10 Member variables cascade for Land CU.

From	To	Description
$dSurface\_runoff\_outflow\_downslope$	$dSurface\_runoff\_inflow\_upslope$	The surface runoff downslope
$dDunnian\_runoff\_outflow\_downslope$	$dDunnian\_runoff\_inflow\_upslope$	The dunnian runoff downslope
$dGroundwater\_outflow\_downslope$	$dGroundwater\_inflow\_upslope$	The groundwater runoff downslope
$dInterflow\_outflow\_downslope$	$dInterflow\_inflow\_upslope$	The soil interflow runoff downslope
$dLitter\_doc\_outflow\_downslope$	$dLitter\_doc\_inflow\_upslope$	The litter doc from downslope
$dSoil\_doc\_outflow\_downslope$	$dSoil\_doc\_inflow\_upslope$	The soil doc from downslope

4. To Stream

   A number of flux terms are cascaded. Note that regardless of types of lateral water flow, they all considered as lateral inflow for a stream reach.

Table 14.11 Member variables cascade for Stream CU.

From	To	Description
$dSurface\_runoff\_outflow\_downslope$	$dDischarge\_inflow\_lateral\_daily$	The surface runoff downslope
$dDunnian\_runoff\_outflow\_downslope$	$dDischarge\_inflow\_lateral\_daily$	The dunnian runoff downslope
$dGroundwater\_outflow\_downslope$	$dDischarge\_inflow\_lateral\_daily$	The groundwater runoff downslope
$dInterflow\_outflow\_downslope$	$dDischarge\_inflow\_lateral\_daily$	The soil interflow runoff downslope
$dLitter\_doc\_outflow\_downslope$	$dDoc\_inflow\_upslope$	The litter doc from downslope
$dSoil\_doc\_outflow\_downslope$	$dDoc\_inflow\_upslope$	The soil soil from downslope

5. To Swale

   Not implemented

14.7.6 From Stream

A Stream CU can only cascaded to the downstream Stream CU. The implementation of stream routing is written in the reach class.

14.7.7 From Swale

This is not implemented.

14.7.8 Auxiliary Function

A number of auxiliary functions are used to assist a few features.

1. Calculate index based on location

```
//=========================================================
//retrieve the column index from the coordinates
//=========================================================
long ecosystem::calculate_index_from_coordinates(long i,
    long j);
```

Details of these functions are presented within the source code.

14.8 Stream

A number of member functions are used to run the stream routing module.

1. Read stream topology.

```
//50=================================================
/*!
Read the stream topology from the input file

\param
\return <ReturnValue>
*/
//50=================================================
int eco3d::read_stream_segment_topology()
```

   The stream topology file describes the spatial relationship (upstream and downstream) between stream segments.

   This file is directly exported from ArcGIS program and no additional edits are required. An example file is provided in Section 22.1.1

2. Create stream topology relationship

```
//50=================================================
/*!
Create the stream segment reach topology vector.
After this function, the vStream_segment_reach will be
    updated.

\param
\return <ReturnValue>
*/
//50=================================================
int eco3d::create_stream_segment_reach_topology()
```

   This function is used to generate the segment reach topology vector.

3. Run stream reach simulation

```
//=======================================================
//run stream module
//=======================================================
int run_stream_reach();
This function is used to run the stream module.
```

14.9 Control

A number of member functions are used to control different model simulation scenarios.

1. Summarize the budget

```
//=======================================================
//summarize system status such as total carbon storage
//=======================================================
int ecosystem::summarize(int iYear, int iDay);
```

   This function is used to summarize a few system state variables.

2. Check whether the system is in steady state or not.

```
//=======================================================
//Check the status of a current simulation run
//=======================================================
int ecosystem::check_steady_state();
```

   This function is used to check whether the system is almost at steady state or not.

14.10 Save

Only one function is used to save model output in current version.

1. Save system status

```
//=======================================================
//save the output variable at each time step
//=======================================================
int save_system_status(int iDay,
int iYear);
```

   This function is used to output the desired variables into output directory at each time step.

14.11 Cleanup

1. Clean up memory

```
//=======================================================
//clean up memory
//=======================================================
int ecosystem::cleanup();
```

This function is used to release variable from memory.

CHAPTER 15

GEOLOGY

15.1 Overview

The geology class mainly deals with geological variables including surface elevation.

15.2 Member Variable

A number of member variables are defined.

Table 15.1 Geological variables.

Name	Default value	Unit	Description
iBoundary	0, 1, 2	unitless	Within boundary or not
dResolution	500	m	Resolution of Column Unit
dArea	500×500	m^2	Area of Column Unit
dAspect	-9999.0	degree	Aspect of Column Unit
dElevation	-9999.0	m	Surface elevation
dSlope	-9999.0	degree	Slope of Column Unit
dLatitude	-9999.0	degree	Latitude of Column Unit

CHAPTER 16

ATMOSPHERE

16.1 Overview

The atmosphere class deals with all processes and variables occurring within the near-surface of Earth. It includes three subclasses: precipitation, radiation, and temperature.

As mentioned earlier, the atmosphere class is independent with the column unit class and it's shared with all types of CUs (Chapter 11).

16.2 Precipitation

16.2.1 Overview

The precipitation class mainly deals with variables related to precipitation event. For example, it classifies precipitation into rain, snow or mixture event based on air temperature.

The precipitation module is developed based on the PRMS model with modifications.

16.2.2 Member Variable

A number of member variables are defined.

Table 16.1 Geological variables.

Name	Default value	Unit	Description
$iPrecipitation\_type$	1, 2, 3, 4	unitless	The type of precipitation
$iFlag\_new\_snow$	0	unitless	Whether there is new snow or not
$iStorm\_type$	1, 2	unitless	The type of storm
$dPrecipitation$	0.0	mm	Total precipitation
$dRain$	0.0	mm	Precipitation in rain form
$dSnow$	0.0	mm	Precipitation in snow form

Four types of precipitation are defined:

1. Rain
2. Snow
3. Mixture
4. No precipitation

16.2.3 Member Function

1. Run

```
//=======================================================
//the headquarter
//=======================================================
int run(int iMonth,
double dPrecipitation,
double dTemperature_max,
double dTemperature_min
);
```

This function is used to start the precipitation module.

2. Calculate rain or snow

```
//=====================================================
//determine the type of precipitation based on temperature
//=====================================================
int calculate_rain_or_snow(double dPrecipitation,
double dTemperature_max,
double dTemperature_min
);
```

This function is used to calculate the partition of rain and snow in a precipitation event.

16.3 Radiation

16.3.1 Overview

The radiation class mainly estimates radiation fluxes including shortwave radiation (SW) and photosynthetically active radiation (PAR).

16.3.2 Member Variable

A number of member variables are defined.

Table 16.2 Radiation variables.

Name	Default value	Unit	Description
$dShortwave\_radiation\_potential$	0.0		Potential SW
$dShortwave\_radiation\_actual$	0	unitless	Actual SW
$dPar$	0.0	unitless	PAR

16.3.3 Member Function

A number of member functions are defined.

1. calculate actual shortwave radiation

```
//=======================================================
//calculate actual shortwave radiation
//=======================================================
double calculate_actual_shortwave_radiation(int iMonth,
double dSlope_radian,
double dTemperature_max
);
```

2. calculate photosynthesis active radiation

```
//=======================================================
//calculate photosynthesis_active_radiation
//=======================================================
double calculate_photosyntheis_active_radiation();
```

3. calculate the potential solar radiation

```
//=======================================================
//calculate the potential solar radiation
//=======================================================
double calculate_potential_shortwave_radiation(int iDay,
double dAspect,
double dLatitude,
double dSlope
);
```

4. head quarter

```
    //=======================================================
    //head quarter
    //=======================================================
    int run(int iDay,
    int iMonth,
    double dAspect,
    double dLatitude,
    double dSlope,
    double dTemperature_max
    );
```

5. calculate potential solar radiation

```
    //=======================================================
    //calculate potential solar radiation
    //=======================================================
    double calculate_potential_solar_radiation_on_surface(
    double dLatitude_radian,
    double dLatitude_difference_radian,
    double dRadiation_hourly,
    double dSolar_declination_radian,
    double dSunrise_time,
    double dSunset_time
    );
```

6. calculate the solar declination

```
    //=======================================================
    //calculate the solar declination
    //=======================================================
    double calculate_solar_declination(int iDay);
```

7. obliquity

```
    //=======================================================
    //obliquity
    //=======================================================
    double calculate_sun_obliquity(int iDay);
```

8. calculate the sunrise and sunset time

```
    //=======================================================
    //calculate the sunrise and sunset time
    //=======================================================
    std::array<double, 2> calculate_sunrise_sunset_time(double
         dLatitude_radian,
    double dSolar_declination_radian
    );
```

16.4 Temperature

16.4.1 Overview

The temperature class mainly calculates several temperature indices

16.4.2 Member Variable

A number of member variables are defined.

Table 16.3 Temperature variables.

Name	Default value	Unit	Description
$iFlag\_transpiration$	0.0	unitless	Potential SW
$dTemperature\_cumulative$	0	kelvin	Actual SW
$dTemperature\_max$	0.0	kelvin	PAR
$dTemperature\_mean$	0.0	kelvin	PAR
$dTemperature\_min$	0.0	kelvin	PAR
$dTemperature\_dewpoint$	0.0	kelvin	PAR
$dVapor\_pressure\_deficit$	0.0	Pascal	PAR
$vTemperature\_mean$	0.0	kelvin	PAR

16.4.3 Member Function

A number of member functions are defined.

1. transpiration status

```
//=======================================================
//
//=======================================================
int calculate_transpiration_status(double
    dTemperature_mean);
```

2. vapor pressure deficit

```
//=======================================================
//
//=======================================================
int calculate_vapor_pressure_deficit(double
    dTemperature_dewpoint,
double dTemperature_mean
);
```

3. run

```
    //=========================================================
    //
    //=========================================================
    int run(double dTemperature_dewpoint,
    double dTemperature_max,
    double dTemperature_min
    );
```

CHAPTER 17

GLACIER

17.1 Overview

Glacier is one type of column type in the ECO3D model (Chapter 11). Glacier CU is charicrized by the glacier coverage on top of the land surface.

The ECO3D model mainly considers the glacier CU dynamics through processes occur associated with glacier (glacier melting, etc.).

A typical glacier CU is composed of several components including landcover and snow.

17.2 Landcover

The landcover class is used to deal with all land cover type related variables and processes.

In general, land cover datasets are used to define land surface type regardless of CU type.

17.2.1 Member Variable

A number of member variables are defined.

Table 17.1 Landcover variables.

Name	Default value	Unit	Description
$iLand\_cover\_type$	1	unitless	The land cover type
$iLand\_cover\_type\_default$	1	unitless	The default land cover type

Sixteen types of land cover are defined based on and land cover datasets:

1. Water
2. Evergreen Needleleaf forest
3. Evergreen Broadleaf forest
4. Deciduous Needlelead forest
5. Deciduous Broadleaf forest
6. Mixed forest
7. Closed shrublands
8. Open shrublands
9. Woody savannas
10. Savannas
11. Grasslands
12. Permanent wetlands
13. Croplands
14. Urban and built-up
15. Croplands/Natural vegetation mosaic
16. Snow and ice
17. Barren or sparsely vegetated

17.2.2 Member Function

A number of member functions are defined.

1. Initialize

```
//=========================================================
//initialize
//=========================================================
int initialize();
```

2. Update

```
//=========================================================
//the update
//=========================================================
int update_parameter(int iLand_cover_type);
```

For glacier CU, the land cover type is usually ice or snow.

17.3 Hydrology Response Unit

The HRU class is used to deal with all HRU related variables and processes.

Hydrologic Response Unit (HRU), the basic computational unit assumed to be homogeneous in hydrologic response to land cover change, is defined through watershed delineation and land cover type. Unlike CU, HRU only describes hydrologic characteristics (the percentage of pervious/impervious surface, etc.).

17.3.1 Member Variable

A number of member variables are defined.

Table 17.2 HRU variables.

Name	Default value	Unit	Description
$iHru\_type$	1	unitless	The type of HRU
$iHru\_type\_default$	1	unitless	The type of default HRU
$dFraction\_impervious$	0	unitless	The type of default HRU
$dFraction\_pervious$	1	unitless	The type of default HRU
$dFraction\_depression$	0.0	unitless	The type of default HRU

17.3.2 Member Function

A number of member functions are defined.

1. Initialize

```
//=======================================================
//initialize
//=======================================================
int initialize(int iColumn_type);
```

2. Update

```
//=======================================================
//the update
//=======================================================
int update_parameter(double dFraction_depression, double
    dFraction_impervious,
double dFraction_pervious);
```

Details of these functions can be found in the source code.

17.4 Snow

17.4.1 Overview

The snow class is used to deal with all snow dynamics related variables (snow cover, snowpack thickness, etc.) and processes (snow accumulation and melting, etc.).

The snow model is developed based on the PRMS model.

17.4.2 Member Variable

A number of variables are defined.

Table 17.3 Snow variables.

Name	Default value	Unit	Description
$iFlag\_melt\_season\_potential$	1	unitless	The type of land cover
$iFlag\_melt\_season\_force$	1	unitless	The type of land cover
$iFlag\_has\_snow$	1	unitless	The type of land cover
$iFlag\_new\_snow$	1	unitless	The type of land cover
$iFlag\_sca\_on\_curve$	1	unitless	The type of land cover
$iFlag\_partly\_reset\_albedo$	1	unitless	The type of land cover
$iFlag\_albebo\_method$	1	unitless	The type of land cover
$iDay\_since\_last\_snow$	1	unitless	The type of land cover
$iDay\_since\_last\_reset$	1	unitless	The type of land cover
$iCount\_above\_zero\_day$	1	unitless	The type of land cover
$dSnow\_cumulative\_save$	1	unitless	The type of land cover
$dAlbedo$	1	unitless	The type of land cover
$dAlbedo_previous$	1	unitless	The type of land cover
$dSca$	1	unitless	The type of land cover
$dSca\_previous$	1	unitless	The type of land cover
$dSnowmelt$	1	unitless	The type of land cover
$dSnowmelt\_day$	1	unitless	The type of land cover
$dSnowmelt\_night$	1	unitless	The type of land cover
$dSnowmelt\_hru$	1	unitless	The type of land cover
$dSwe$	1	unitless	The type of land cover
$dSwe\_density$	1	unitless	The type of land cover
$dSwe\_depth$	1	unitless	The type of land cover
$dSwe\_heat\_deficit$	1	unitless	The type of land cover
$dSwe\_ice$	1	unitless	The type of land cover
$dSwe\_max$	1	unitless	The type of land cover
$dSwe\_plus$	1	unitless	The type of land cover
$dSwe\_previous$	1	unitless	The type of land cover
$dSwe\_sub\_max$	1	unitless	The type of land cover
$dSwe\_temperature$	1	unitless	The type of land cover
$dSwe\_temperature\_celsius$	1	unitless	The type of land cover
$dSwe\_threshold$	1	unitless	The type of land cover
$dSwe\_water$	1	unitless	The type of land cover
dEt_snow	1	unitless	The type of land cover
dEt_snow_hru	1	unitless	The type of land cover
$dTemperature\_surface\_snow$	1	unitless	The type of land cover

17.4.3 Member Function

A number of member functions are defined.

1. Initialize

```
//=========================================================
//initialize
//=========================================================
int initialize(int iColumn_type);
```

2. Update

```
//=========================================================
//the update
//=========================================================
int update_parameter();
```

Details of these functions can be found in the source code.

17.5 Glacier

By definition, there is glacier coverage on this type of column unit.

17.6 Surface Runoff

17.6.1 Overview

The surface runoff, specifically, the overland surface runoff, is simulated using the algorithm from PRMS.

17.6.2 Member Variable

A number of member variables are defined.

Table 17.4 Member variables of surface runoff class.

Name	Default value	Unit	Description
$dET\_impervious$	0	m	ET from impervious surface
$dET\_impervious\_hru$	0	m	HRU averaged ET from impervious surface
$dFraction\_impervious$	0	unitless	fraction of impervious surface
$dRunoff\_impervious$	0	m	Surface from impervious surface
$dRunoff\_impervious\_hru$	0	m	HRU averaged runoff from impervious surface
$dStorage\_impervious$	0	m	Storage from impervious surface
$dStorage\_impervious\_hru$	0	m	HRU averaged storage from impervious surface
$dFraction\_pervious$	0	unitless	fraction of pervious surface
$dRunoff\_pervious$	0	m	Surface from pervious surface
$dRunoff\_pervious\_hur$	0	m	HRU averaged runoff from pervious surface
$dSurface\_runoff\_inflow$	0	m	total surface runoff inflow
$dSurface\_runoff\_outflow$	0	m	total surface runoff outflow
$dSurface\_runoff\_storage$	0	m	total surface runoff storage
$dSurface\_runoff\_inflow_{e}xternal$	0	m	total surface runoff inflow
$dSurface\_runoff\_inflow_{u}pslope$	0	m	total surface runoff inflow
$dSurface\_runoff\_outflow_{d}ownslope$	0	m	total surface runoff outflow
$dSurface\_runoff\_outflow_{i}nfiltration$	0	m	total surface runoff outflow
$dSurface\_runoff\_hru$	0	m	total surface runoff outflow
$dET\_surface$	0	m	total surface runoff outflow
$dET\_surface\_hru$	0	m	total surface runoff outflow
$dContribution\_area$	0	m	total surface runoff outflow
$dContribution\_area\_max$	0	m	total surface runoff outflow
$dContribution\_area\_min$	0	m	total surface runoff outflow

17.6.3 Member Function

A number of member functions are defined.

1. Initialize

```
int initialize(int iColumn_type);
```

2. calculate impervious surface runoff

```
int calculate_impervious_surface_runoff(double
    dEt_available_in,
double dSca_in,
double dWater_available_in);
```

3. calculate impervious surface ET

```
int calculate_impervious_surface_evapotranspiration(double
    dEt_available_in,
double dSca_in,
double dWater_available_in);
```

4. calculate impervious surface storage

```
int calculate_impervious_surface_storage(double
    dWater_available_in);
```

5. calculate pervious surface runoff

```
int calculate_pervious_surface_runoff(int iFlag_old_snow, int
    iHru_type,
int iPrecipitation_type,
double dSoil_moisture_in,
double dSoil_moisture_max_in,
double dStorage_recharge_zone_in,
double dStorage_recharge_zone_max_in,
double dTemperature_mean_in,
double dWater_available_pervious_in);
```

6. calculate variable source area

```
double calculate_variable_source_area(double dSoil_recharge_in
    ,
double dSoil_recharge_max_in);
```

7. update parameter

```
int update_parameter(int iLand_cover_type, int iSoil_type);
```

8. run

```
int run(int iFlag_old_snow, int iFlag_has_snow, int iHru_type,
int iLand_cover_type, int iPrecipitation_type,
double dET_available_hru_in,
double dRain_net_in,
double dSnow_net_in,
double dSCA_in,
double dSnowmelt_in,
double dSoil_moisture_in,
double dSoil_moisture_max_in,
double dSoil_temperature_in,
double dStorage_recharge_zone_in,
double dStorage_recharge_zone_max);
```

Details of these functions can be found in the source code.

17.7 Infiltration

17.7.1 Overview

The infiltration class is used to calculate surface infiltration using a continuum equation.

17.7.2 Member Variable

A number of member variables are defined.

Table 17.5 Member variables of surface runoff class.

Name	Default value	Unit	Description
$dInfiltration$	0	m	ET from impervious surface
$dInfiltration\_hru$	0	m	HRU averaged ET from impervious surface
$dSurface\_runoff\_excess$	0	unitless	fraction of impervious surface
$dSnow\_infiltration\_max\_base$	0	m	Surface from impervious surface
$dSnow\_infiltration\_max$	0	m	HRU averaged runoff from impervious surface

17.7.3 Member Function

A number of member functions are defined.

1. calculate infiltration

```
int calculate_infiltration(
    int iFlag_old_snow, int iHru_type, int
        iPrecipitation_type,
    double dFraction_pervious_in,
    double dRunoff_pervious_in,
    double dSoil_moisture_in,
    double dSoil_moisture_max_in,
    double dWater_available_pervious_in);
```

2. calculate excess infiltration

```
int calculate_excess_snowmelt_infiltration(double
    dSoil_moisture_in,
double dSoil_moisture_max_in);
```

3. Initialize

```
int initialize(int iColumn_type);
```

4. Update parameter

```
int update_parameter(int iLand_cover_type, int
    iSoil_type);
```

5. Run

```
int run(int iFlag_old_snow,
int iHru_type,
int iPrecipitation_type,
double dFraction_pervious_in,
double dRunoff_pervious_in,
double dSoil_moisture_in,
double dSoil_moisture_max_in,
double dSoil_temperature_in,
double dWater_available_pervious_in);
```

Details of these functions can be found in the source code.

CHAPTER 18

LAKE

18.1 Overview

In our first release, lake is not fully implemented. It is used as a place holder.

18.2 HRU

18.3 Landcover

18.4 Snow

18.5 Groundwater

CHAPTER 19

LAND

19.1 Overview

Land Column Unit type is the major component within the ECO3D model.

Most water cyle and carbon cycle processes are simulated for the Land CU.

19.2 Evapotranspiration

19.2.1 Overview

The evapotranspiration class is currently developed based on algorithm from PRMS.

19.2.2 Member Variable

A number of member variables are defined.

Table 19.1 Member variables of surface runoff class.

Name	Default value	Unit	Description
$dCoefficient\_jh$	0	m	Coefficient
$dPet$	0	m	total potential et
$dPet\_cm$	0	m	ET in centimeter
$dLambda$	0	m	latent heat
$dPet\_hru$	0	m	HEU averaged ET

19.2.3 Member Function

A number of member functions are defined.

1. initialize

```
int initialize();
```

2. update parameter

```
int update_parameter();
```

3. calculate vaporization latent heat

```
int calculate_vaporization_latent_heat(double
    dTemperature_mean_in);
```

4. calculate jh coef

```
int calculate_jh_coef(double dElevation_in,
double dVapor_pressure_deficit);
```

5. JH method

```
int calculate_potential_evapotranspiration_jh(int
    iMonth,
double dElevation_in,
double dShortwave_in,
double dTemperature_mean_in,
double dVapor_pressure_deficit_in);
```

6. PT method

```
        int calculate_potential_evapotranspiration_pt(int
            iMonth,
      double dElevation_in,
      double dShortwave_radiation_in,
      double dTemperature_mean_in,
      double dTemperature_mean_yesterday_in);
```

7. Run

```
        int run(int iMonth,
      double dElevation_in,
      double dShortwave_radiation_actual_in,
      double dTemperature_mean_in,
      double dVapor_pressure_deficit_in);
```

Details of these functions can be found in the source code.

19.3 Vegetation

19.3.1 Overview

Currently, vegetation coverage only exists on land CU. The vegetation class deals with all vegetation related processes.

19.3.1.1 Member Variable

A number of member variables are defined.

19.3.1.2 Member Function

A number of member functions are defined.

19.3.2 Canopy

19.3.3 Overview

Several important processes are simulated at the canopy level, which includes photosynthesis and interception.

19.3.3.1 Member Variable

A number of member variables are defined.

Table 19.2 Canopy variables.

Name	Default value	Unit	Description
$dCarbon$	1	m	Carbon storage
$dNitrogen$	1	m	Carbon storage
$dLai$	1	m	Carbon storage
$dScalar\_foliage$	1	m	Carbon storage
$dkleafc$	1	m	Carbon storage
$dcov$	1	m	Carbon storage

19.3.3.2 Member Function

A number of member functions are defined.

1. Update parameter

```
int update_parameter(int iFlag_vegetation_previous, int
    iVegetation_type);
```

2. Calculate foliage scalar

```
int calculate_foliage_scalar(int iVegetation_type);
```

3. Initialize

```
int initialize(int iColumn_type, int iVegetation_type);
```

4. Run

```
int run(int iMonth, int iPrecipitation_type, double
    dArea_in, double dPET_in,
double dLai_in, double dRain_in, double dSnow_in,
double dFpar_in, // climate inputs
double dPAR_in,  // estimated from radiation module
double dTemperature_mean_in,
double dTemperature_min_in,       // climate input
double dVapor_pressure_deficit_in // esti
);
```

5. Update

```
int update();
```

19.3.4 Photosynthesis

19.3.4.1 Overview

The photosynthesis model is developed based on the MODIS GPP algorithm.
The GPP is calculated by:

$$GPP = \epsilon \times APAR \tag{19.1}$$

$$\epsilon = \epsilon_{max} \times TMIN_{scalar} \times VPD_{scalar} \tag{19.2}$$

$$APAR = IPAR \times FPAR \tag{19.3}$$

$$IPAR = SW \times 0.45 \tag{19.4}$$

19.3.4.2 Member Variable A number of member variables are defined.

Table 19.3 Canopy variables.

Name	Default value	Unit	Description
$iVegetation\_type$	1	m	Carbon storage
$dFpar$	1	m	Carbon storage
$dScalar\_temperature$	1	m	Carbon storage
$dScalar\_vpd$	1	m	Carbon storage
$dGpp$	1	m	Carbon storage
$dGpp\_gram$	1	m	Carbon storage
$dEpsilon\_max$	1	m	Carbon storage
$dTmin\_max$	1	m	Carbon storage
$dTmin\_min$	1	m	Carbon storage
$dVpd\_max$	1	m	Carbon storage
$dVpd\_min$	1	m	Carbon storage

19.3.4.3 Member Function A number of member functions are defined.

1. Calculate daily GPP

```
int calculate_daily_gpp(double dFpar_in, double dIpar_in,
    double dTemperature_min_in,
double dVapor_pressure_defici_int);
```

2. epsilon

```
double calculate_epsilon(double dEpsilon_max_in, double
    dTemperature_min_in,
double dVapor_pressure_deficit_in);
```

3. Temperature

```
double calculate_temperature_scalar(double dTemperature_min_in
    );
```

4. Vpd

```
double calculate_vpd_scalar(double dVapor_pressure_deficit_in)
    ;
```

5. Initialize

```
int initialize(int iColumn_type, int iVegetation_type);
```

6. Update parameter

```
int update_parameter(int iFlag_vegetation_previous, int
    iVegetation_type);
```

7. Run

```
int run(double dFpar_in, double dPar_in, double
    dTemperature_min_in,
double dVapor_pressure_deficit_in);
```

19.3.5 Interception

19.3.5.1 Overview

The interception class is a new module developed based on the original interception module in PRMS.

19.3.5.2 Member Variable

19.3.5.3 Member Function

19.3.5.4 Litterfall

19.3.5.5 Overview

The litterfall module is developed based on litterfall module in TEM.

19.3.5.6 Member Variable

19.3.5.7 Member Function

19.3.6 Stem

19.3.7 Root

19.4 Litter

19.4.1 Overview

In ECO3D, the little is considered as a pool similar to vegetation and soil.

In solid phase, chemical component including carbon and nitrogen are simulated using litter decompose model.

In liquid phase, dissolved organic/inorganic carbon are simulated as well.

In gaseous phase, nothing is yet simulated.

19.4.2 Decomposition

19.4.3 Carbon

For the carbon component, the litter carbon dynamics are simulated using a two-pool model.

First, the total carbon pool receive incoming carbon flux from litterfall and dissolved organic carbon. It also contributes outgoing flux in several forms:

1. humus
2. dissolved organic carbon to downslope
3. dissolved organic carbon through infiltration
4. respiration by litter microbe

Second, the total carbon is classified into the undecomposed and decomposed pools.

The conversion from undecomposed to decomposed is modeled using a classical exponent approach.

19.4.4 Dissolved Organic Carbon

Dissolved Organic Carbon (DOC) is an important component in the ECO3D model.

In the litter model, DOC is modeled using similar approach used in soil DOC transport.

Even though DOC is expressed using concentration, the mass units are used for transport purpose because the continuum equation is based on mass balance.

19.4.5 Nitrogen

Nitrogen is currently not considered.

19.5 Surface Runoff

19.6 HRU

19.7 Landcover

19.8 Snow

19.9 Soil

19.9.1 Respiration

19.10 Groundwater

CHAPTER 20

STREAM

20.1 Overview

Steam is one type of column type in the ECO3D model (Chapter 11). Stream CU is defined through watershed delineation process.

The ECO3D model mainly considers processes associated with stream networks (stream routing, etc.).

As usual, a stream CU is composed of several components including HRU and land cover.

In traditional hydrological models, stream networks are usually represented by a number of connected stream segments. In the ECO3D model, each stream segment is further discretized into a number of connected stream reach, which has the same spatial resolution with other grid cells. In this way, we can simulated stream related processes and variables (DOC/DIC, etc.) for individual stream reach.

20.2 HRU

The HRU type of a stream CU is stream, which is defined through watershed delineation process.

20.3 Landcover

The land cover type of a stream CU is usually water.

20.4 Groundwater

For a stream CU, the groundwater component is used to consider the groundwater and stream water interactions.

20.5 Reach

20.5.1 Overview

The reach class is used to simulated all processes occured within or associated with a stream reach.

20.5.2 Theoretical Basis

In the ECO3D model, a stream reach is the smallest unit within which the mass balanced stream routing and solute transport are simulated.

For the stream routing algorithm, I used the same routing algorithm used in the PRMS model. However, there are several improvements.

1. Fisst, it is no longer stream segment based, it stream reach based, which means the new approach could significantly improve the capability considering stream geometry.

2. Second, at high spatial-temporal resolution, the travel time of each stream reach is much shorter (¡ 10 minutes, etc.). Therefore, the corresponding time step for stream routing is modified and the simulation can produce semi-real time estimates.

3. Third, stream routing now fully supports external source/sink terms, which means that any reach can receive external inflow (upstream inflow, pipeline, etc.).

   To accept external inflow, the input text files should have a required format. An exmaple of this text-based file is illustrated

```
segment, 01
nreach, 2
nvariable, 2
1, 2000, 1, 1, 0.0, 0.0
1, 2000, 1, 2, 1.0, 0.1
1, 2000, 1, 3, 0.0, 0.0
1, 2000, 1, 4, 0.0, 0.0
1, 2000, 1, 5, 0.0, 0.0
2, 2000, 1, 1, 0.0, 0.0
2, 2000, 1, 2, 1.0, 0.1
2, 2000, 1, 3, 0.0, 0.0
2, 2000, 1, 4, 0.0, 0.0
2, 2000, 1, 5, 0.0, 0.0
```

In this example, the text file contains external inflow for a stream segment (ID = 01), and this segment has 2 reaches. For each reach, 2 variables are provided, the first one is the stream discharge, and the second one could be DOC concentration. The units of the inflow must be the same with the member variables (Table 20.1). After the header, time series of inflow are listed for each stream reach. In this case, inflow of 5 days data are prepared.

20.5.3 Member Variable

A number of member variables are used to store the state variables of a stream reach.

Table 20.1: Member variables of the reach class.

Name	Default value	Unit	Description
$iColumn\_type$	1	unitless	The type of stream
$iTime\_step$	1	unitless	The default time step, daily
$dLength$	1	meter	Length of stream reach
$dSlope$	1	degree	Slope of stream reach bed
$dTravel\_time$	1	day	The actual travel time
$dTravel\_time\_default$	1	day	The default travel time
$dTravel\_time\_minute$	1	minute	The actual travel time in minute
$dTravel\_time\_minute\_default$	1	minute	The default travel time in minute
$dInflow\_daily$	1	m^3d^{-1}	The total daily inflow
$dOutflow\_daily$	1	m^3d^{-1}	The total outflow
$dInflow\_lateral\_daily$	1	m^3d^{-1}	The total daily inflow from lateral flow
$dInflow\_upstream\_daily$	1	m^3d^{-1}	The total daily inflow from upstream
$dStorage\_daily$	1	m^3	The total storage in this reach
$dInflow\_minute$	1	m^3d^{-1}	The total inflow in minute
$dOutflow\_minute$	1	m^3d^{-1}	The total outflow in minute
$dInflow\_lateral\_minute$	1	m^3d^{-1}	The total inflow from lateral flow in minute
$dInflow\_upstream\_minute$	1	m^3d^{-1}	The total inflow from upstream in minute
$dStorage\_minute$	1	m^3	The total storage in minute
$dInflow\_previous\_minute$	1	m^3d^{-1}	The previous total inflow in minute
$dOutflow\_previous\_minute$	1	m^3d^{-1}	The previous total outflow in minute
$dInflow\_second$	1	m^3d^{-1}	The total inflow in second
$dOutflow\_second$	1	m^3d^{-1}	The total outflow in second
$dInflow\_lateral\_second$	1	m^3d^{-1}	The total inflow from lateral flow in second
$dInflow\_upstream\_second$	1	m^3d^{-1}	The total outflow from upstream in second
$dStorage\_second$	1	m^3	The total storage in second

$dDoc\_daily$	1	kg	The daily total DOC in the reach
$dDoc\_inflow\_daily$	1	$kg\,d^{-1}$	The daily total DOC inflow
$dDoc\_upstream\_daily$	1	$kg\,d^{-1}$	The daily DOC from upstream
$dDoc\_lateral\_daily$	1	$kg\,d^{-1}$	The daily DOC from lateral flow
$dDoc\_downstream\_daily$	1	$kg\,d^{-1}$	The daily DOC to downstream
$dDoc\_concentration\_daily$	1	$kg\,d^{-1}$	The daily DOC concentration in the reach
$dDoc\_minute$	1	$kg\,d^{-1}$	The total DOC in minute
$dDoc\_inflow\_minute$	1	$kg\,d^{-1}$	The total DOC inflow in minute
$dDoc\_upstream\_minute$	1	$kg\,d^{-1}$	The DOC from upstream in minute
$dDoc\_lateral\_minute$	1	$kg\,d^{-1}$	The DOC from lateral flow in minute
$dDoc\_downstream\_minute$	1	$kg\,d^{-1}$	The DOC to downstream in minute
$dDoc\_concentration\_minute$	1	$kg\,d^{-1}$	The DOC concentration in minute
$dInflow\_external\_daily$	1	m^3d^{-1}	The daily inflow from external source
$dDoc\_concentration\_external\_daily$	1	$kg\,m^{-3}d^{-1}$	The DOC concentration in the external source
k	1	unitless	
x	1	unitless	
$c0$	1	unitless	
$c1$	1	unitless	
$c2$	1	unitless	
$cGeology$	1	unitless	The geology object
$cHru$	1	unitless	The HRU object
$cGroundwater$	1	unitless	The groundwater object
$cLandcover$	1	unitless	The land cover type object

20.5.4 Member Function

A number of member functions are defined.

1. Initialize

```
//=====================================================
//initialize
//=====================================================
int initialize(int iColumn_type);
```

This function is used to initialize the reach class member variables.

2.

```
//=====================================================
//the update
//=====================================================
int update_parameter();
```

This function is used to update some reach parameters. It is unlikely a stream reach will change significantly over a short period of time. But I keep this capability.

3.
```
//=======================================================
//the update
//=======================================================
int calculate_reach_discharge(int iMinute);
```

This function is used to calculate the reach discharge.

4.
```
//=======================================================
//the update
//=======================================================
int calculate_reach_dissolved_organic_carbon_concentration
    (int iMinute);
```

5.
```
//=======================================================
//the update
//=======================================================
int calculate_reach_travel_time();
```

6.
```
//=======================================================
//the update
//=======================================================
int run();
```

Details of these functions can be found in the source code.

20.6 Segment

20.6.1 Overview

Even thought the ECO3D model maily focuses on stream simulation at reach level, it also provide statistical analyses at segment level.

20.6.2 Member Variable

20.6.3 Member Function

CHAPTER 21

ISSUE

1. Hydraulic and hydrologic compatibility

   Because I have used several algorithms from PRMS, which use hydrologic methods for infiltration, soil zone interflow, surface runoff and stream routing, they may be upgraded with hydraulic methods in the newer version.

   To guarantee the compatibility between these two types of systems, I have to consider the framework of the model structure.

   First, regardless of hydraulic or hydrologic methods, the continuum equation will be used to ensure mass balance.

   $$\frac{dStorage}{dt} = flow_{incoming} - flow_{outgoing} \tag{21.1}$$

   Second, due to nature of flow system, a temporal resolution is required for numerical simulation. Even though in some systems (PRMS, etc), a daily time step is used for lots of hydrologic processes, the actual flow system (overland surface runoff, etc) may be different, especially when the spatial resolution is high. In the current ECO3D model, the temporal resolution for overland surface runoff is daily and the spatial resolution is $500.0\,m$, which can be related through Manning's equation.

   In the future development, the temporal resolution for overland surface runoff should be minute-base with consideration of spatial resolution. An assessment can be conducted using the Manning equation.

2.

PART IV

MODEL DEMONSTRATION

CHAPTER 22

EXAMPLE

22.1 Inputs

```
workspace_data /home/liao46/workspace/cplus/ecosystem/resource/
    data
workspace_out /scratch/rice/l/liao46/snyder/model/ecosystem/
    output
workspace_dewpoint /scratch/rice/l/liao46/snyder/product/gsod/
    dewpoint/huc819040507/extract
workspace_fpar /scratch/rice/l/liao46/snyder/product/modis/
    mcd15a3/fpar/huc819040507/extract
workspace_lai /scratch/rice/l/liao46/snyder/product/modis/
    mcd15a3/lai/huc819040507/extract
workspace_prec /scratch/rice/l/liao46/snyder/product/ncdc/prec/
    huc819040507/extract
workspace_tmax /scratch/rice/l/liao46/snyder/product/ncdc/tmax/
    huc819040507/extract
workspace_tmin /scratch/rice/l/liao46/snyder/product/ncdc/tmin/
    huc819040507/extract
workspace_land_cover /scratch/rice/l/liao46/snyder/product/modis
    /mcd12q1/landcover/huc819040507/extract
```

Workspace

Raster dataset

```
aspect aspect.dat
boundary boundary.dat
column_index column_index.dat
dem dem.dat
hru_type hru_type.dat
latitude latitude.dat
slope slope.dat
soil_type soil_type.dat
stream_reach stream_reach.dat
stream_segment stream_segment.dat
stream_order stream_order.dat
```

Text file

```
segment_topology segment_topology.txt
starlog starlog.txt
header header.hdr
```

Controls

```
start_year 1990
end_year 1995
start_month 1
end_month 12
start_day 1
end_day 31

steady_state_flag 1
transient_flag 1

initial_soil_carbon soil_carbon.dat
initial_vegetation_carbon vegetation_carbon.dat
initial_litter_carbon litter_carbon.dat
```

Output

22.1.1 Stream Topology

```
FID_,HydroID,GridID,NextDownID
,1,42,6
,2,32,6
,3,49,7
,4,50,7
,5,43,8
,6,41,8
,7,40,12
,8,39,12
,9,54,13
,10,53,13
,11,38,14
,12,35,14
```

```
  ,13,36,15
  ,14,37,15
  ,15,34,18
  ,16,29,20
  ,17,28,20
  ,18,27,22
  ,19,11,22
  ,20,10,23
  ,21,7,23
  ,22,5,24
  ,23,9,26
  ,24,4,26
  ,25,13,30
  ,26,8,30
  ,27,31,32
  ,28,30,32
  ,29,16,34
  ,30,12,34
  ,31,48,35
  ,32,26,35
  ,33,22,36
  ,34,15,36
  ,35,24,37
  ,36,21,37
  ,37,23,0
```

PART V

TOOL

CHAPTER 23

TOOL

23.1 Overview

To use the ECO3D model, a number of tools and programs are required/recommended to facilitate the pre/post-processes.

23.2 ArcGIS

ArcGIS is a geographic information system for working with maps and geographic information. In this document, ArcGIS has beed used to generate several important input data (stream topology file, etc.).

23.3 Watershed Delineation

23.3.1 Stream Reach Topology

PART VI

APPENDIX

APPENDIX A

FAQ

A.1 What is the difference between Column Unit and Hydrologic Response Unit?

Answer: CU and HRU have a lot of common. However, there are also several differences. First, CU emphases more on the object-oriented programming approach. It is the basic object of the ECO3D model. Meanwhile, HRU only emphases on the hydrologic characteristics.

Also, CU not only consider hydrologic processes, but also other processes in the vertical profile.

APPENDIX B

UNIT

Index