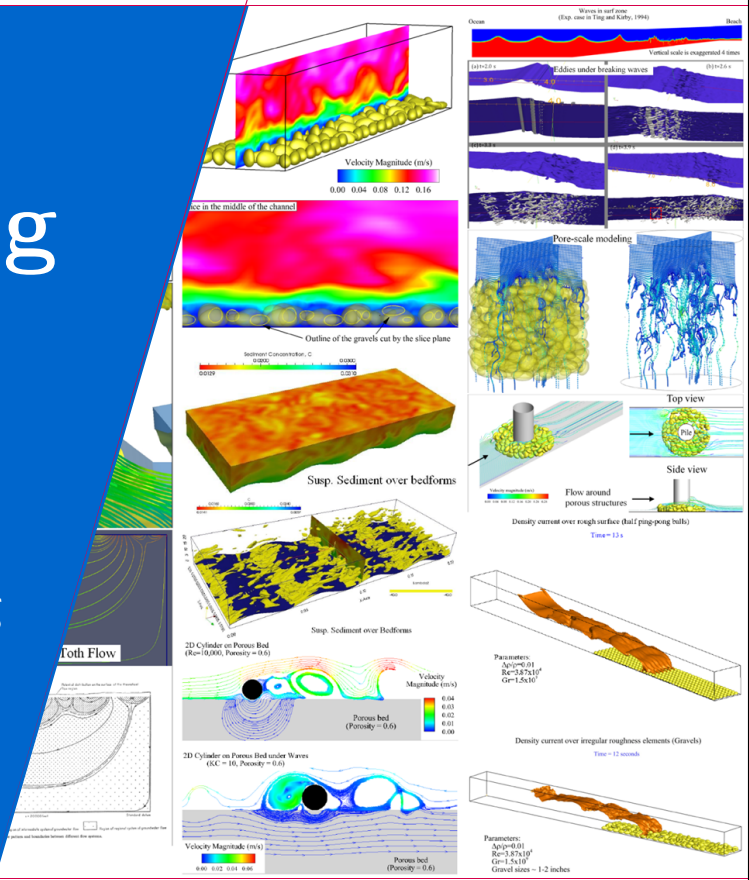
PENNSTATE

1855

CE597D Special Topics, Fall, 2014

Chapter 5: Programming and customization

Xiaofeng Liu, Ph.D., P.E.
Assistant Professor
Department of Civil and Environmental Engineering
Pennsylvania State University
xliu@engr.psu.edu



What will be covered in this chapter?

This chapter is for intermediate and advanced level users:

- Develop a new solver
- Develop a new boundary condition
- Develop a new utility tool
- Debugging

Outline

The structure of OpenFOAM

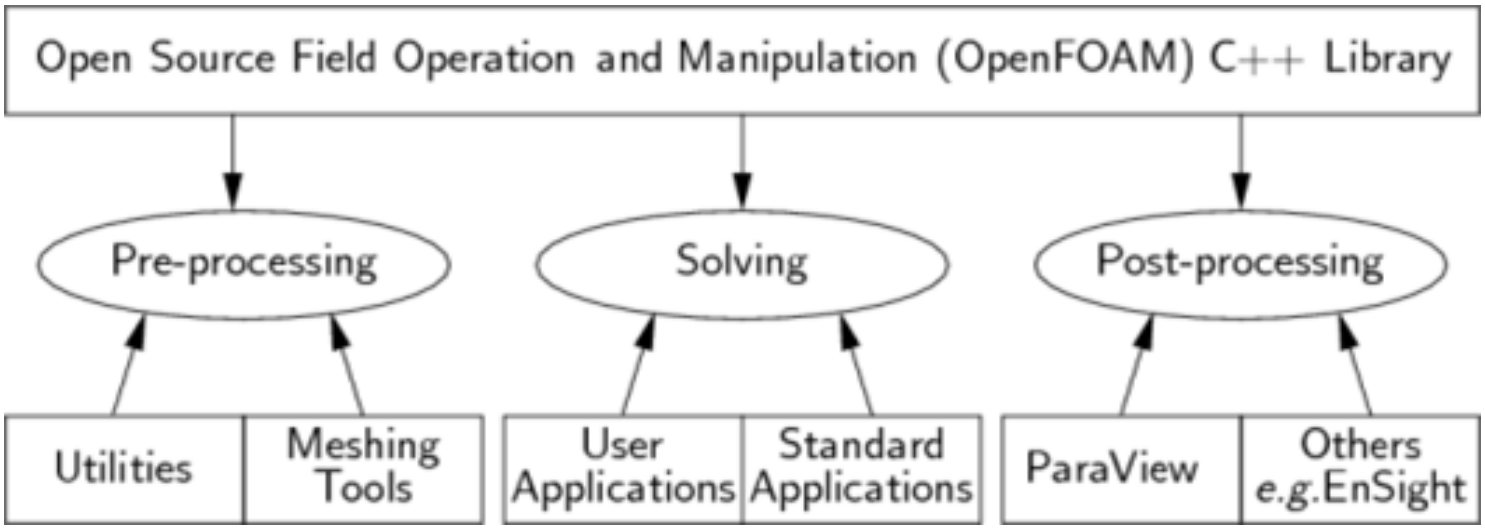


Figure: Structure of the OpenFOAM® platform

Development on OpenFOAM

Before you do anything, search or ask!

- Do not re-invent the wheel!
- The OpenFOAM® user community is getting so big and the applications are very broad
- There might be someone has done similar work
- Try search the web by using key words (including OpenFOAM® of course)
- Or ask the community (the CFD-online forum, OF extend project, etc.)

Development on OpenFOAM

Code organization and compilation:

- Best place to start is the OpenFOAM® code itself
- Usually we find the closest code and make a copy (with a different name of course).
  - For example, to write a new solver, just copy a similar existing solver
  - Change the name of the files and resulting executable
  - Modify code and compile
  - Use the new solver
- OpenFOAM® uses `wmake` to organize the compilation process
- Detailed instructions:

  `http://www.openfoam.org/docs/user/compiling-applications.php`

Development on OpenFOAM

Usage of `wmake`:

- `files`: specify what C++ source files to compile, the result name, and where to put it.
- The targeted result can be executables (solver and utility tools)

```
newApp.C

EXE = $(FOAM_USER_APPBIN)/newApp
```

or library

```
newLib.C

LIB = $(FOAM_USER_LIBBIN)/libnewLib
```

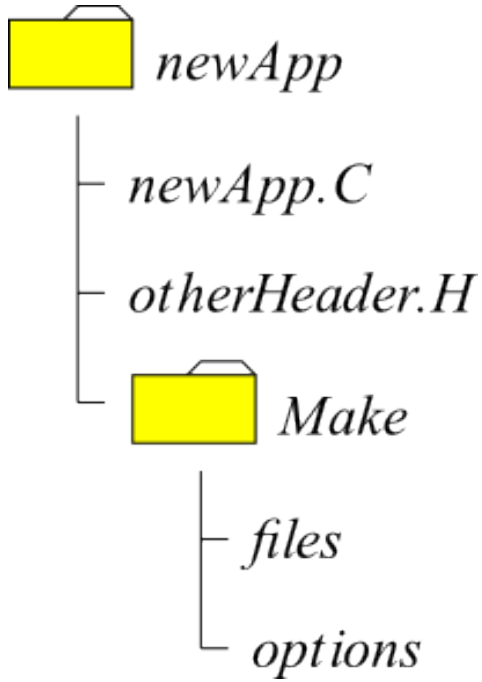


Figure: wmake

Development on OpenFOAM

Usage of `wmake`:

- `options`: specify the paths to the header files to be included and the libraries to be linked.

```
EXE_INC = \
    -I<directoryPath1> \
    -I<directoryPath2> \
                       \
    -I<directoryPathN>

EXE_LIBS = \
    -L<libraryPath1> \
    -L<libraryPath2> \
                     \
    -L<libraryPathN> \
    -l<library1>     \
    -l<library2>     \
                     \
    -l<libraryN>
```

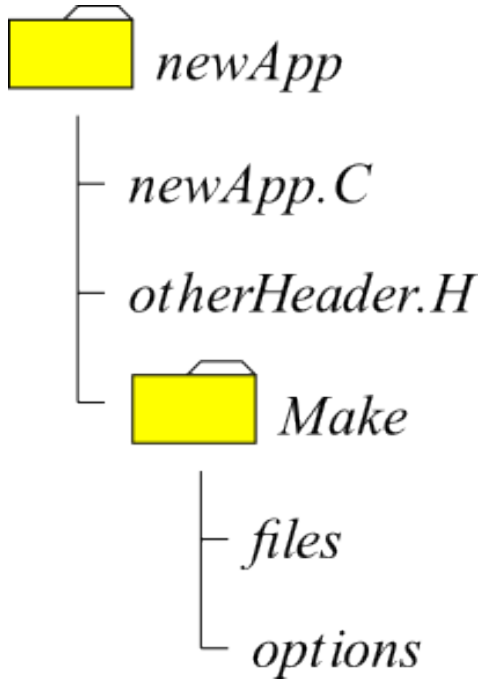


Figure: wmake

Development on OpenFOAM

Usage of `wmake`:

- To compile: just type `wmake`
- In OpenFOAM® , `wmake` uses a lot of environmental variables, such as `WM_PROJECT`, `FOAM_USER_LIBBIN`,etc
- `wmake` generates a lot of intermediate files.
- It is a good idea to cleanup using the `wclean` command before compilation

Develop a new solver

- OpenFOAM® comes with a lot of pre-defined solvers
- However, they might not meet your needs.
- Examples:
  - Your problem is totally new and has not been implemented in OpenFOAM® .
  - Your problem is not totally new but you need to modify existing solver.
  - You need to add more physics to the problem, say add scalar transport to the `pisoFoam` solver.

Develop a new solver

New solver examples in porous media and groundwater flows:

- Saturated (Darcy law, diffusion equation, linear, easy)
- Unsaturated (Richardson equation, non-linear, not easy)

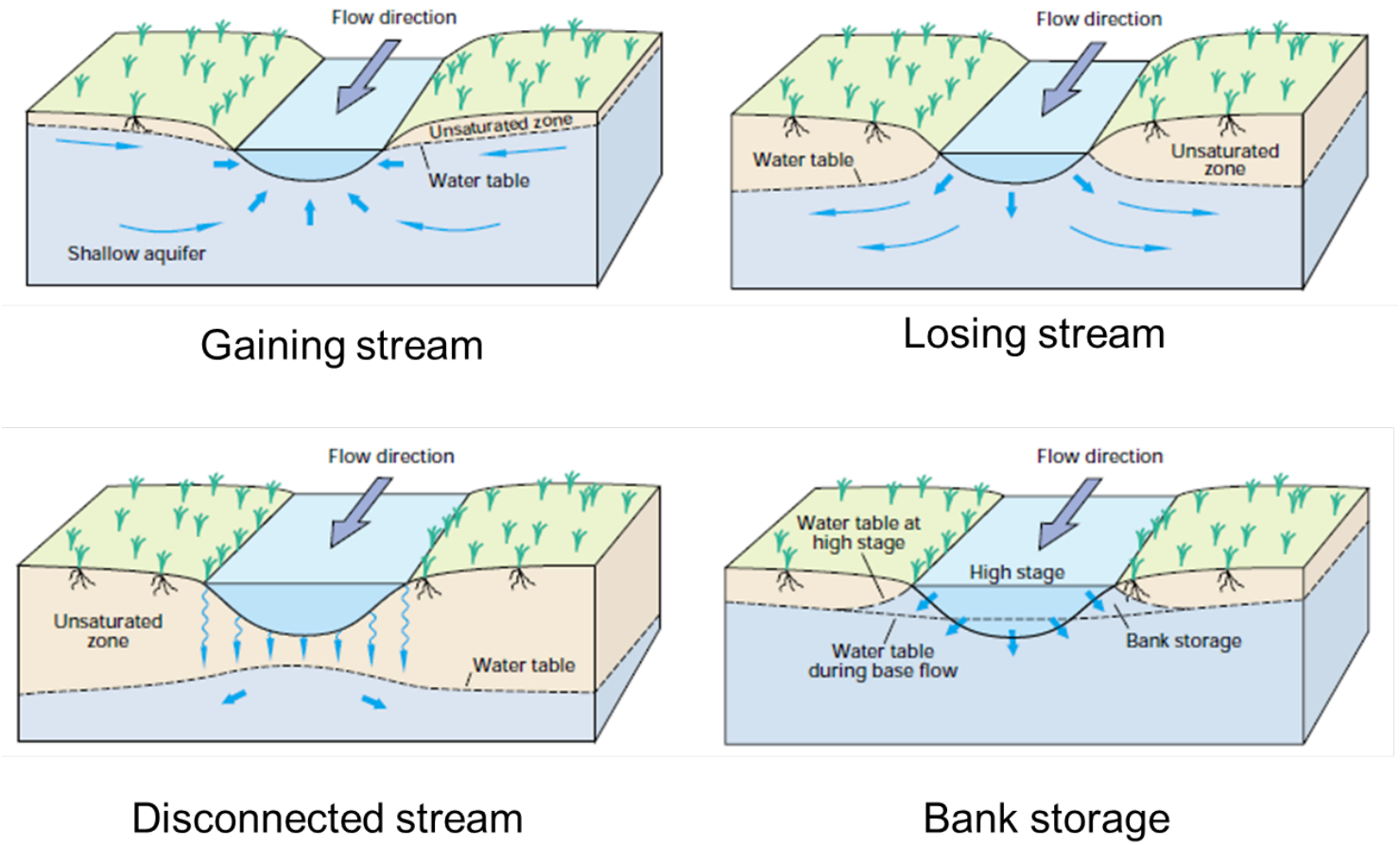


Figure: Surface water-groundwater interactions

Develop a new solver

Saturated groundwater flow:

- Darcy law, diffusion equation, linear, easy
- This governing equation is a simple heat equation. The solution of which is very easily implemented in OpenFOAM® using tensor notations.
- The new solver: `groundWaterFoam`

$$S_s \frac{\partial h}{\partial t} = K \nabla^2 h + Q$$

where h is the pressure head, Ss is the specific storage coefficient, K is hydraulic conductivity, Q is source/sink.

Develop a new solver

The new solver: groundWaterFoam:

```
    while (simple.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        for (int nonOrth=0; nonOrth<=simple.nNonOrthCorr(); nonOrth++)
        {
            solve
            (
                fvm::ddt(h)
              - fvm::laplacian(K/Ss, h)
             ==
                Q_o_Ss
            );
        }

#       include "write.H"

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
```

Develop a new solver

The new solver: `groundWaterFoam`

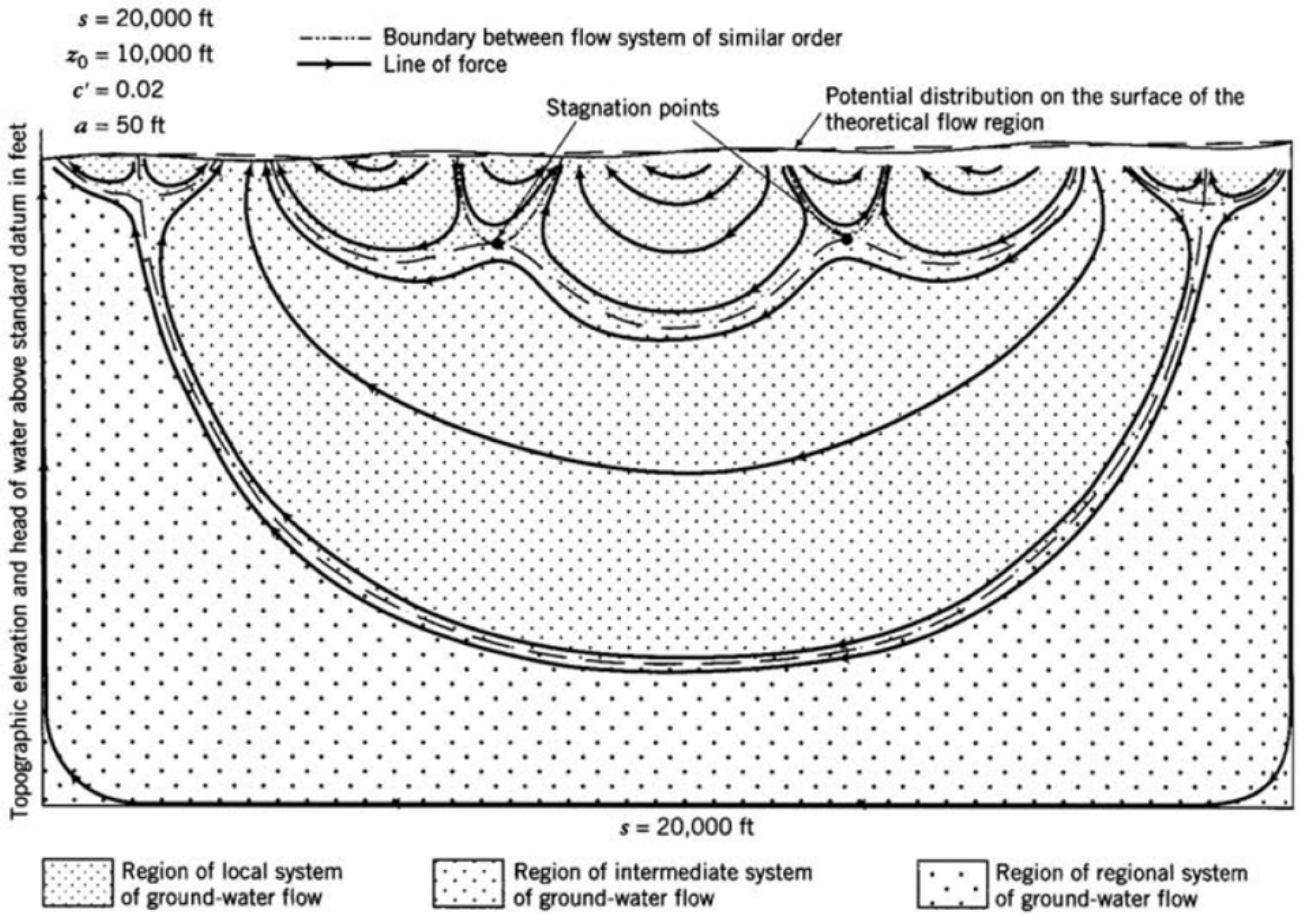


Figure: Scheme of Toth flow (Toth, 1963)

Develop a new solver

The top boundary condition for h used `groovyBC`.

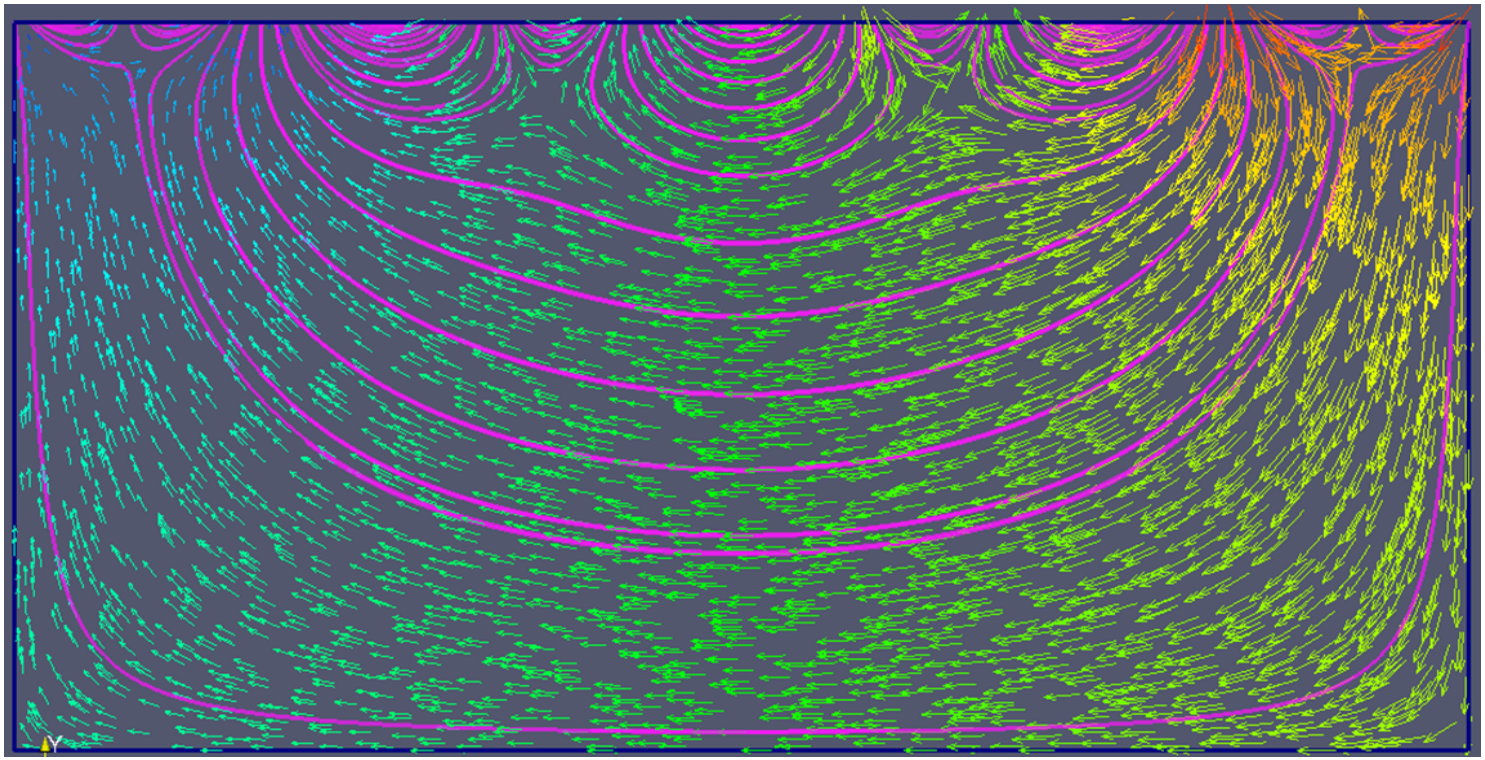
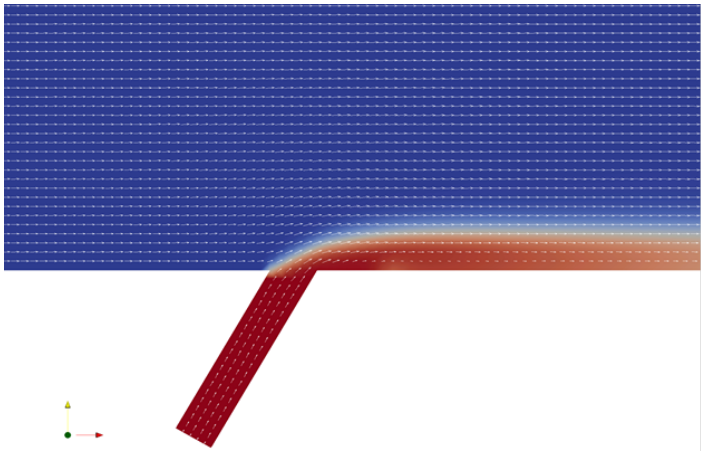


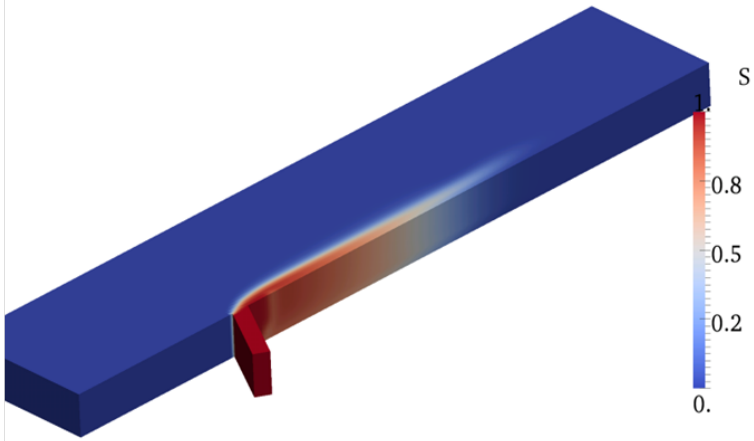
Figure: Simulation result of Toth flow (Toth, 1963)

Develop a new solver

In the tutorial, you will learn how to adapt the `pisoFoam` solver for adding a scalar transport equation:

$$\frac{\partial S}{\partial t} + \nabla \cdot (\mathbf{u}S) = \nabla \cdot (D \nabla S) \tag{1}$$





Develop a new B.C.

- OpenFOAM® also comes with a lot of pre-defined boundary conditions
- However, they might not meet your needs.
- Examples:
  - You need to specify velocity distribution at the inlet.
  - You need to specify a boundary condition change with time
  - Other boundary conditions needs to be hard-coded
- Again, before you do, search and ask!

Develop a new B.C.

- There are at least two ways to do it
  - Add the implement of new BC directly into your solver
  - Write a new library implementing the new BC and dynamically load it at run time
- Pros and cons of both methods

Develop a new B.C.

Option 1: Implement the new BC into your solver:

- The generic BCs in OpenFOAM are located in

  `$FOAM_SRC/finiteVolume/fields/fvPatchFields/`

- Find the one closest to your needs
- Copy its files to the solverąś code directory
- Change the files according to your needs
- Modify `Make/files` and `Make/options` and instruct the compiler to compile these new BC files
- Recompile the solver

Develop a new B.C.

Option 1: Implement the new BC into your solver:

- Demonstration: Modify the `oscillatingFixedValue` BC

Develop a new B.C.

Option 2: Write a new library for the BC

- The generic BCs in OpenFOAM are located in

  `$FOAM_SRC/finiteVolume/fields/fvPatchFields/`

- Find the one closest to your needs
- Copy its files to the solverąś code directory
- Change the files according to your needs
- Modify `Make/files` and `Make/options` and instruct the compiler to compile these new BC files
- Compile these new BC files to a library

Develop a new B.C.

Option 2: Write a new library for the BC

- Demonstration: Create a new library for the new `oscillatingFixedValue` BC

Develop a new B.C.

A closer look at the boundary condition code

- It has several .H and .C files
- First, it defines a unique name by

  `Typename(ąřmyOscillatingFixedValueąś)`

  in the `myOscillatingFixedValueFvPatchField.H` head file
- This head file also defines the class

```
template<class Type>
class myOscillatingFixedValueFvPatchField
:   public fixedValueFvPatchField<Type>
```

- It is templated, i.e., good for Type scalar, vector, and tensor fields.

Develop a new B.C.

A closer look at the boundary condition code

- It has three private data

```
Field<Type> refValue_;
scalar amplitude_;
scalar frequency_;
```

- Each boundary condition class has an `updateCoeffs` member function, which is called every time step to update the boundary condition

```
void myOscillatingFixedValueFvPatchField<Type>::updateCoeffs()
```

- You need to change here according to your needs

Develop a new utility tool

- OpenFOAM® comes with a lot of pre-defined tools (in `applications/utilities`)
- However, they might not meet your needs.
- Examples:
  - Your need to convert OF result to certain format for visualization
  - You need to calculate flow rate across some surface
  - You need to calculate the total kinetic energy in the simulation domain
  - ...

Develop a new utility tool

Implement a new utility tool:

- The generic tools in OpenFOAM® are located in `$FOAM_APP/utilities/`
- Find the one closest to your needs
- Copy its files to a new directory
- Change the files according to your needs
- Modify `Make/files` and instruct the compiler to compile these new files
- Recompile the tool

Develop a new utility tool

- Demonstration: new utility tool `oneFieldToTecplot`

Debugging

- What if something is not right with your code?
- You need to do debugging.
- Ways to do it:
  - Print information to the screen/file
    - Use `Info << ąřsomethingąś << A_variable << endl;`
    - `Info` is a pre-defined object of class `messageStream` which handles messages
    - Most of classes in OpenFOAM® implement the << operator.
    - So `Info << U << endl;` will dump everything about velocity U field to the screen.

Debugging

- Ways to do it:
  - Use `DebugSwitches`
    - Defined in file `$WM_PROJECT_DIR/etc/controlDict`
    - It has the following information

```
DebugSwitches
{
        Analytical      0;
        ...
        fvVectorMatrix    0;
        ...
}
```

    - Switch 0 turns off the debug information; 1, 2, 3, etc. turns on different levels of debug information
    - You don't need to run the `debug` version of OpenFOAM® to use `DebugSwitches`
    - Each simulaiton case can have its own definition of `DebugSwitches` in `system/controlDictfile`

Debugging

- Ways to do it:
  - Use `gdb` (Gnu Debugger)
    - It is the professional way to do debugging
    - It is doable
    - But I have not done it so far.
    - You need to recompile a debug version of OpenFOAM® by

      `export WM_COMPILE_OPTION=Debug`

      in file `$WM_PROJECT_DIR/etc/bashrc`
    - Then you need to load the executable into `gdb` to do things like stepping, break, print variables, etc.

Questions?