CHALMERS UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF APPLIED MECHANICS

DIVISION OF FLUID DYNAMICS

CFD WITH OPENSOURCE SOFTWARE, ASSIGNMENT 3

Tutorial - shallowWaterFoam

Developed for OpenFOAM-1.7.x

Author:
JOHAN PILQVIST

Peer reviewed by:
FIRSTNAME LASTNAME
HÅKAN NILSSON

October 17, 2010

Introduction

- This tutorial will aim towards explaining the fundamentals of the solver shallowWaterFoam. The main focus will be to analyze the code functionality and the implementation of the *shallow water equations*.
- To give some context to the code functionality we will follow the standard case squareBump

The shallow water equations

The *shallow water equations* are a set of differential equations that describe the motion of an incompressible fluid within a domain whose depth is considered to be shallow compared with the radius of the Earth.

Assuming that the fluid is incompressible and that the effects of vertical shear of the horizontal velocity is negligble, the equations can be derived by depth-integrating the continuity and Navier-Stokes equations.

The shallow water equations

The momentum and continuity equations for the shallow water equations read

$$\frac{\partial}{\partial t}(h\mathbf{u}) + \nabla \cdot (h\mathbf{u}^T\mathbf{u}) + f \times h\mathbf{u} = -|\mathbf{g}|h\nabla(h + h_0) + \tau^w - \tau^b \tag{1}$$

$$\frac{\partial}{\partial t}(h + h_0) + \nabla \cdot (h\mathbf{u}) = 0 \tag{2}$$

where

- h is the mean surface height,
- $\mathbf{u}$ is the velocity vector,
- $f = (2\Omega \cdot \hat{\mathbf{g}})\hat{\mathbf{g}}$ is the *Coriolis force* (depending on the angular rotation rate of the Earth, Ω, and $\hat{\mathbf{g}}$ being the normal vector of gravity),
- h_0 is the deviation from the mean surface height and
- τ^w and τ^b are the wind and bottom stresses respectively.

The shallow water equations

In shallowWaterFoam, the wind and bottom stresses are assumed to be zero. Also, the surface velocity flux is defined as

$$\phi_v = \phi/h = \{\phi = h\mathbf{u} \cdot \hat{\mathbf{n}}\} = \mathbf{u} \cdot \hat{\mathbf{n}} \tag{3}$$

where $\hat{\mathbf{n}}$ is the cell face area vector. Equation 1 then reduces to

$$\frac{\partial}{\partial t}(h\mathbf{u}) + \nabla \cdot (h\phi_v\mathbf{u}) + f \times h\mathbf{u} = -|\mathbf{g}|h\nabla(h + h_0) \tag{4}$$

Equations 4 are the equations that are solved in shallowWaterFoam.

Running a case

Now we are going to apply the shallowWaterFoam solver on the squareBump case located in the tutorials folder. Before setting up this case, copy the entire folder from the tutorials section into your own run directory;

```
cd $FOAM_RUN
cp -r $FOAM_TUTORIALS/incompressible/shallowWaterFoam/squareBump .
cd squareBump
```

Geometry

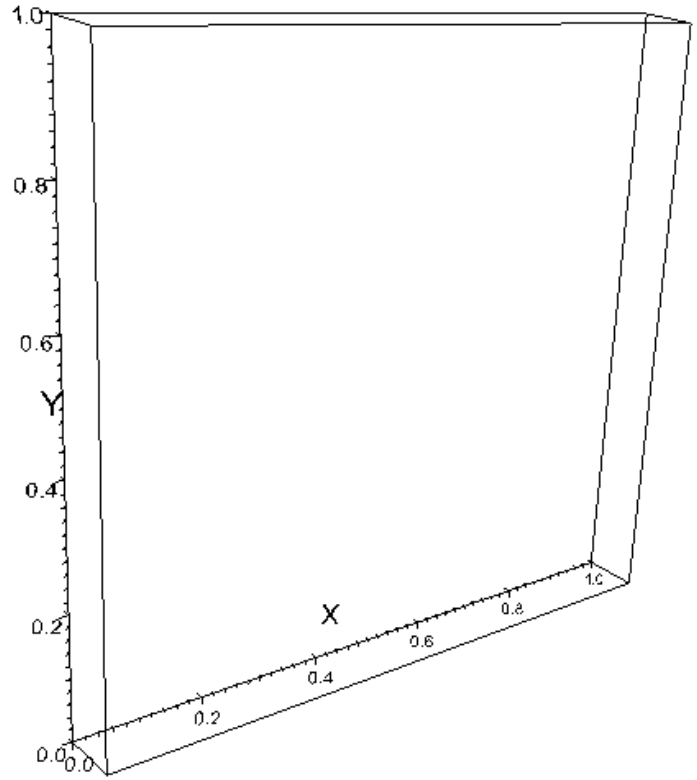

Figure 1: Geometry of the shallowWaterFoam tutorial case squareBump.

The geometry in the squareBump case consists of a single hexahedron block with a 1×1 meter base and a depth of 0.1 meter (Figure 1).

Meshing

To create a mesh we use the command

`blockMesh`

This generates a mesh following the descriptions in the dictionary blockMeshDict located in the subdirectory constant/polyMesh.

From this file we can conclude that

- the mesh is to be built up by a single hexahedron block of 20×20 cells
- the mesh is to be axially equidistant.

blockMeshDict also divides the boundary areas of the domain into different patches. These are later used in order to define the boundaries in the file boundary, which is also located in the subdirectory constant/polyMesh. The mesh could now be viewed in ParaView and it will look as in Figure 2.

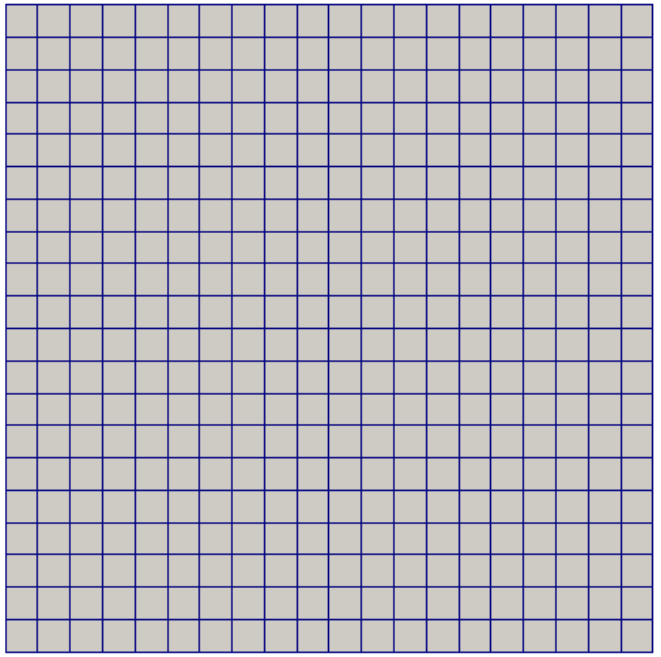

Figure 2: The default mesh.

Initial and boundary conditions

In squareBump there are four initial conditions that need to be defined;

- the *mean* surface height h,
- the *deviation* from the mean surface height h_0,
- the *free-surface* height $h_{total}(= h+h_0)$ and
- the velocity vector field $\mathbf{u}$.

The initial velocity vector field is uniformly distributed from the far left and onto the internal field of the domain, with a magnitude of 0.1 m/s. Moreover, the four midmost cells has an initial *mean* surface height of 0.009 meters (the darker square in the middle of Figure 3) simulating a quadratic obstacle, or a *square bump*.

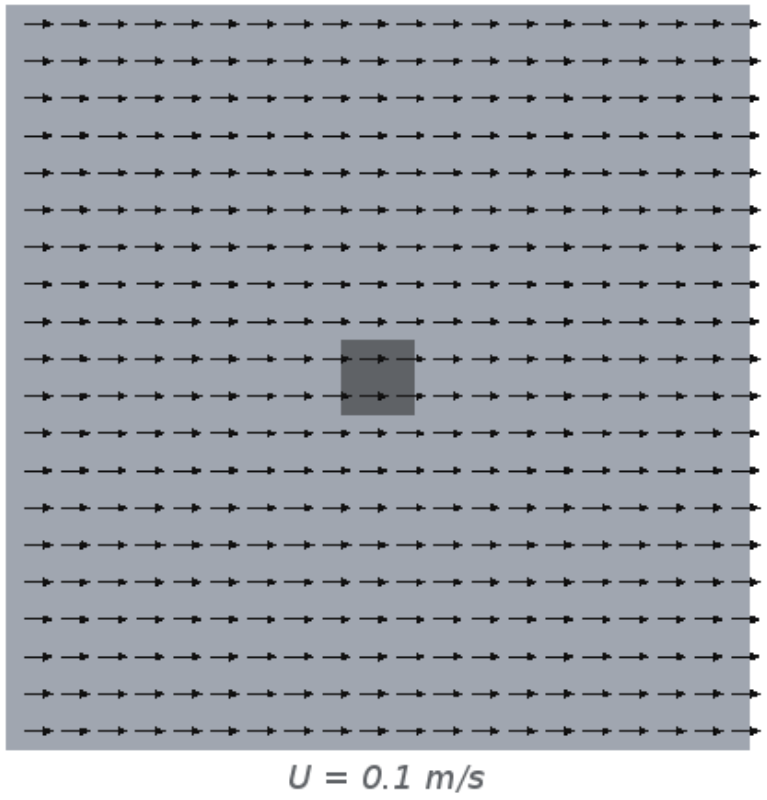

Figure 3: Visualization of the initial velocity vector field, $\mathbf{u} = (0.1\ 0\ 0)$ m/s, and the non-uniform initial surface height (denoted by different shades of gray).

Initial and boundary conditions

- h_{total} is given a uniform value of 0.01 m.
- Since h is not uniformly distributed, but h_{total} is, the equality $h_{total} = h + h_0$ then calls for h_0 to balance h_{total}. To do this h_0 is given the value 0 for all cells except the four midmost ones, using a non-uniform list of scalars (similar to the declaration of h).

The initial conditions related to the actual case should all be located in the subdirectory 0. Unfortunately though, for some reason the file h0 is by default located in the wrong subdirectory, namely constant. To correct this mistake we need to move this file to the correct location (i.e. the subdirectory 0). This is done using the following command;

```
mv constant/h0 0/
```

Initial and boundary conditions

The *point*/*cell* values of the initial conditions now need to be applied onto the mesh as *fields*. This is done using the utility setFields, simply by typing

```
setFields
```

The dictionary for this utility, setFieldsDict, is located in the sub-directory system.

Worth noting is that had we not moved the file h0 to the correct directory, this command could not have been executed. We would instead have received an error message stating that setFields.C was unable to locate the file h0.

Physical properties

The *shallow water equations* are dependent of the gravitational force and the rotation of the Earth. These physical properties are defined in the file gravitationalProperties located in the subfolder constant.

From this file we conduct that

- the gravitational force $g = 9.81$ m/s^2
- the angular rotation rate of the Earth $\Omega = 7.292 \cdot 10^{-5}$ s^{-1}.

Controlling the simulation

Before running a case we need to set some preferences for controlling the simulation process, as well as the output of the results. These settings are done in the files located in the subfolder system.

From controlDict we can, for example, deduce that

- the squareBump case is to be simulated from 0 to 100 seconds (remember that we have indeed defined all our initial conditions for $t = 0$ s, i.e. in the 0 directory).
- the timestep, Δt, is 0.1 second.
- the writeInterval is set to 1, meaning that we will write to our results every *one* seconds. Hence, we will *solve* for a thousand timesteps of which we will *save* the data for every tenth timestep.

Controlling the simulation

From fvSchemes we can see that

- the time discretization is done using Crank-Nicholson. It is however done using a θ value of 0.9, meaning that it is very close to being fully implicit.
- the divergence scheme is explicitly specified.

From fvSolution we can see what types of solver algorithms that are used to solve the different equation systems. Furthermore, it includes some input arguments to the PISO controls;

Table 1: The input arguments to the PISO loop defined in fvSolution.

PISO argument	Value	Description
nOuterCorrectors	3	Input value to the outer for loop solving the entire equation system
nCorrectors	1	Input value to the correction loop for ϕ (i.e. the face flux field)
nNonOrthogonalCorrectors	0	Input value to the correction loop for h (i.e. the mean surface height)
momentumPredictor	yes	Activation of momentum predictor

Running the simulation

Now the case has been thoroughly setup and is ready to be solved. Go to the terminal prompt and type

```
shallowWaterFoam 2>&1 | tee log
```

Post-processing

To post-process the case, open ParaView;

```
paraFoam
```

Some observations:

- In the beginning the velocity is largely influenced by the differences in surface height and continuity forces the flow to disrupt the uniform flow conditions set for $t = 0$ s.
- The coupling between $\mathbf{u}$ and h in the momentum equations causes the flow to initiate a wave motion.
- As time passes, the flow is decelerated and the wave motion consequently fades out.

The solver

The path to the solver is

```
$FOAM_APP/solvers/incompressible/shallowWaterFoam/shallowWaterFoam.C
```

The solver starts off with inclusion of fvCFD.H, which subsequently contains even more inclusions. These are needed in order to define and/or declare different classes, types, functions and variables used in shallowWaterFoam.C and its subfiles.

The solver

The first function main also starts off with some inclusions;

- setRootCase.H makes sure that shallowWaterFoam is executed in a valid directory. If not, it returns the message "`FOAM FATAL IO ERROR`".
- createTime.H defines the time properties according to the settings in the controlDict file.
- createMesh.H reads the mesh generated by blockMesh for which the equations are to be solved.
- readGravitationalAcceleration.H reads the gravitational constant, g, and the angular rotation rate of the Earth, Ω, to later be used in the equations.
- createFields.H reads the initial values for the different scalar and vector fields (e.g. h, $\mathbf{u}$ etc.) defined in the current case. It also calculates the $Coriolis$ force and names it F.

The solver

The inclusion of CourantNo.H in the beginning of the while loop is needed to calculate the $Courant$ (or CFL) number used to evaluate the time step. Since we ran the squareBump case using the command

```
shallowWaterFoam 2>&1 | tee log
```

we can easily confirm that $CFL \leq 1$ at all times by having a look in the file log.

The PISO arguments are read into the solver code via the inclusion of readPISOControls.H. Through this file some of the arguments experience a slight change of name, or rather become abbreviated. Still, they are easily recognized in the solver code as

- nOuterCorr,
- nCorr and
- nNonOrthCorr.

The solver

Some observations about the PISO controls:

- the momentum equation system is solved *three* times for each time step
- the correction of ϕ and h is done *once* every time the momentum equation system is being solved.

What may be noted is that the for loop concerning the non-orthogonal grids, i.e

```
for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
```

will indeed be executed even though nNonOrthogonalCorrectors was given the value 0 in the PISO controls.

The case uses momentumPredictor;

- usually incorporated to generate a good initial guess for the PISO loop in order to speed up the convergence.
- actually slows down the default squareBump case.

The solver

The solver shallowWaterFoam has been written so that it allows for the user to disregard the rotation of the Earth. Since rotating was set to true in gravitationalProperties, the momentum equations were solved in their entirety (i.e. also accounting for the Earth's rotation).

The left-hand side[1] of the momentum equation system (named hUEqn in the code) is defined as

```
fvVectorMatrix hUEqn
(
    fvm::ddt(hU)
  + fvm::div(phiv, hU)
);
```

where ddt denotes the time-derivative $\partial/\partial t$ and div denotes the divergence (i.e. $\nabla\cdot$). The operators ddt and div use the discretization schemes that were defined in fvSchemes.

[1] Except the *Coriolis* part which, as previously mentioned, is optional to take into consideration

The solver

As an attempt to get faster convergence, the shallowWaterFoam solver uses under-relaxation to solve the hU equation system. This is done using the command

```
hUEqn.relax();
```

The full hU equation system is now defined and solved as follows (provided that the rotation of the Earth is accounted for);

```
hUEqn + (F ^ hU) == -magg*h*fvc::grad(h + h0)
```

where F is the *Coriolis* force, magg denotes |g| and grad denotes the gradient operator (∇). Just as ddt and div the operator grad uses the discretization scheme defined in fvSchemes.