Tutorial for OpenFOAM course: "The buoyantFoam solver"

Margarita Sass-Tisovskaya

April 12, 2008

The *"buoyantFoam"* solver is a transient solver for buoyant, turbulent flow of compressible fluids used for ventilation and heat-transfer.

In this tutorial we will present the test case that help us to understand the main idea of the *bouyantFoam* solver, also we will briefly describe how one can create a wedge geometry and a new thermophysical model. For that purpose, we will consider a small example, that contains the introduction to the problem being solved, setup of the initial/boundary conditions and the physical properties required. In addition, we will also discuss the equations and models being used.

The chosen solution domain is the portion of a original domain due to the symmetry plane conditions. The method of meshing is *blockMesh*; the mesh can be simply viewed in the *polyMesh* directory.

Example for the buoyantFoam solver

An illustrative example of how to use the *buoyantFoam solver* we consider the fluid part of TIG welding.

In TIG welding, a tungsten electrode heats the metal that is being welded and the gas (most commonly Argon) that protects the weld puddle from airborne contaminants. In Fig. 1 you can see the typical sketch of the TIG (Tungsten Inert Gas) welding.

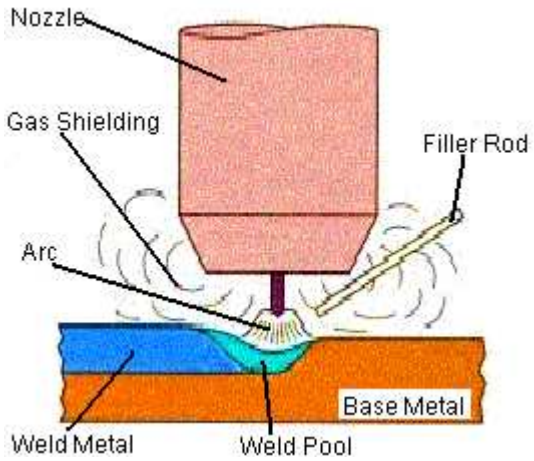


Figure 1: A typical sketch of the TIG welding (www.weldingengineer.com)

In this example the electromagnetic part of the real physical process and the material droplet are skipped. In addition, the weld material is considered to be cooled. The flow of the shielding gas and temperature distribution are considered in the TIG welding process by using the standard *buoyantFoam solver*. The hot argon gas is injected with high velocity into the domain.

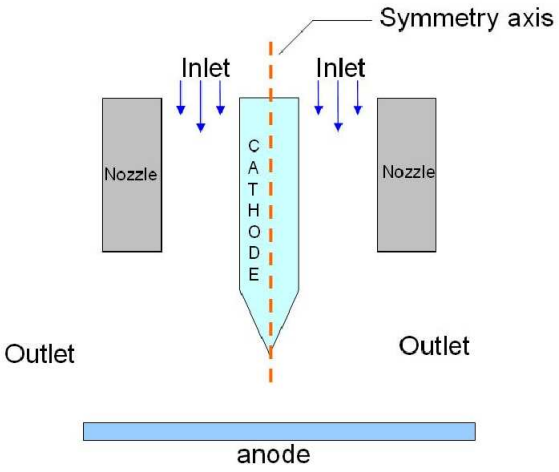


Figure 2: The sketch of the simplified version of TIG welding

For the simplicity, we will set a much lower temperature on the tip of the cathode than in a real case to be able use the constant transport coefficient.

Governing equations

The following equations are used in *boyantFoam solver* and in our problem:

- Continuity equation

  The continuity equation for the compressible fluids is presented by the following equation (*OpenFoam-1.4/OpenFOAM /OpenFOAM-1.4 /src /finiteVolume /cfdTools /compressible /rhoEqn.h*):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \tag{1}$$

  where ρ is density and U is velocity.

- Momentum equations

  The momentum equations for *buoyantFoam* are placed in the *OpenFOAM /OpenFOAM-1.4 /applications /solvers /heatTransfer /buoyantFoam /UEqn.H* file and reads:

$$\frac{\partial(\rho U)}{\partial t} + \nabla \cdot (\rho U \cdot U) - \nabla \cdot [\mu_{eff}(\nabla U + \nabla U^T) - \frac{2}{3}\mu_{eff}(\nabla \cdot U)I] = -\nabla p_d - \nabla \rho g_h \tag{2}$$

  where p_d is dynamic pressure; μ_{eff} is the effective viscosity.

  The effective viscosity is presented as :

$$\mu_{eff} = \mu_{laminar} + \mu_{turbulent}$$

  where $\mu_{laminar}$ is laminar kinematic viscosity and $\mu_{turbulent}$ is turbulent viscosity.

  The equation for dynamic pressure is solved in *OpenFOAM/OpenFOAM-1.4/applications/solvers/heatTransfer/buoyantFoam/pEqn.H* and reads:

$$\frac{\partial \psi p_d}{\partial t} + \frac{\partial \psi}{\partial t} \cdot p_{ref} + \frac{\partial \psi \rho}{\partial t} \cdot + \nabla(\rho U) - \Delta(\rho U) \cdot p_d = 0 \tag{3}$$

  where $\psi = \frac{1}{RT}$ (s^2/m^2) - compressibility, R is the gas constant.

- Energy equation

  The energy equation is represented in terms of the internal enthalpy (*OpenFOAM/OpenFOAM-1.4/applications/solvers/heatTransfer/buoyantFoam/hEqn.H*):

$$\frac{\partial(\rho h)}{\partial t} + \nabla \cdot (\rho U) - \nabla \cdot (\alpha_{eff} \nabla h) = \frac{\partial p}{\partial t} + U \cdot \nabla p \tag{4}$$

  where α_{eff} - thermal diffusivity and is given by

$$\alpha_{eff} = \alpha_{laminar} + \alpha_{turbulent}$$

  where $\alpha_{laminar}$ is laminar thermal diffusivity and $\alpha_{turbulent}$ is turbulent thermal diffusivity.

The temperature is represented by following expression (*OpenFOAM/OpenFOAM-1.4/src/thermophysicalModels/specie/thermo/specieThermo/specieThermoI.H*):

$$T = T_{old} - \frac{H(T_{old}) - h}{C_p}$$

where T_{old} is the temperature at the previous time step, C_p heat capacity and

$$H(T_{old}) = \frac{C_p T + H_f}{W}$$

where H_f reference enthalpy, W molecular weight.

The total pressure p is :

$$p = p_d + \rho g_h + p_{ref}$$

where p_d is dynamic pressure, ρg_h static pressure, g_h gravity and p_{ref} reference pressure (atmospheric).

- Ideal gas

  The ideal gas equation is described by following expression (*OpenFOAM/OpenFOAM-1.4/src/thermophysicalModels/basic/basicThermo.H*):

$$\rho = pRT \tag{5}$$

  where R is the gas constant.

Create the case in OpenFOAM:

The pre-processing in OpenFOAM can be performed using *FoamX* or by editing files by hand. Most OpenFoam users choose to edit files by hand because the I/O uses a dictionary format with keywords with sufficient meaning to be understood by even the least experienced users. *FoamX* is really a layer that interprets the entries and presents them in a GUI. To use *FoamX* you should simply type in the command line

```
FoamX
```

Then choose create the *new case* and select the *"buoyantFoam"* solver. Lets name our *new case "buoyantFoamTIG"*. The use of *FoamX* is described in more detail in Chapter 5 in the User Guide.

After *"buoyantFoamTIG" case* has veeb created, one should be able to see the directory named *"buoyantFoamTIG"*. This directory contains "time" directory 0, *constant* and *system* directories.

Geometry

The domain is 2 dimensional and axis symmetric. The symmetry line is placed along the cathode. To create our mesh we should go to *constant/polyMesh/* and open file *blockMeshDict*. In this file we create the wedge shape mesh. Note that the symmetry axis should be lying along the x-axis.

First step is to add the *vertices*:

```
vertices
(
    (0 0 0)
    (0 0.4999  -0.0109)
    (0 4.9988  -0.1091)
```

```
    (0 16.3960 -0.3578)
    (2 16.3960 -0.3578)
    (2 4.9988  -0.1091)
    (2 0.4999   -0.0109)
    (2 0  0)
    (3.9 1.5996 -0.0349)
    (3.9 4.9988 -0.1091)
    (3.9 16.3960 -0.3578)
    (5 4.9988 -0.1091)
    (5 1.5996 -0.0349)
    (10 4.9988 -0.1091)
    (10 1.5996 -0.0349)
    (0 0.4999 0.0109)
    (0 4.9988 0.1091)
    (0 16.3960 0.3578)
    (2 16.3960 0.3578)
    (2 4.9988 0.1091)
    (2 0.4999 0.0109)
    (3.9 1.5996 0.0349)
    (3.9 4.9988 0.1091)
    (3.9 16.3960 0.3578)
    (5 16.3960 0.3578)
    (5 4.9988 0.1091)
    (5 1.5996 0.0349)
    (10 4.9988 0.1091)
    (10 1.5996 0.0349)

);
```

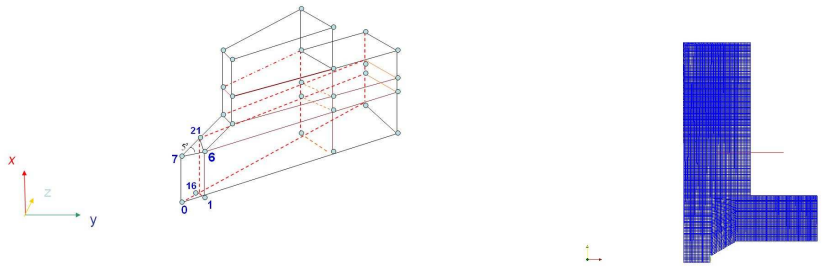
For 2 dimensional axi-symmetric cases, the geometry is specified as a wedge with an angle of 5^o degrees. The next step is to create the *blocks*. In order to create such blocks we should collapse some vertices. Thus, the block numbering becomes

```
hex (0 7 6 1 0 7 21 16) (20 5 1) simpleGrading (1 1 1)
```

This is followed by definition of the patches. The symmetry axis shoud be named *symmetryPlane axis*. The front and back patches will have type *wedge*.

Save your file *blockMeshDict* and type in the command line *blockMesh*, then *checkMesh*. Make sure that the orientation of the vertices and the patches is correct, otherwise *blockMesh* and *checkMesh* will produce an error message.

In order to visualize the mesh, simply type in the command line *paraFoam* . *buoyantFoamTIG* and as a result you should get



(a) Block stucture of the mesh.

(b) Mesh for *"buoyantFoamTIG"*

Figure 3: Computational domain.

For a more detailed explanation see Chapter 6.3.3 in the User Guide.

Boundary and Initial conditions

To set up the boundary conditions, one should edit the *boundary* file. There you will see the names of your boundaries and their physical type . For example

```
rightOutlet
   {
       type            patch;
       physicalType    pressureOutlet;
       startFace       18225;
       nFaces          50;
   }
```

In *0* directory we will set up our values for the boundary.

Nozzle	cathode Tip	cathode wall	anode	Inlet	Outlet
grad(T)=0 WALL	T=5000 K WALL	grad(T)=0 WALL	grad(T)=0 WALL	T=300 K PrabolicInlet($U_x^{max} = 3m/s$)	grad(T)=0 PressureOutlet

Thermophysical properties

The *termophysicalProperties* dictionary is read by *buoyantFoam* solver that uses (in createFields.H) *thermophysical* model library *basicThermo* (*OpenFOAM/OpenFOAM-1.4/src/thermophysicalModels/basic/*). There is a dictionary entry called *thermoType* which specifies the complete thermophysical model that is used in the simulation. The thermophysical modeling starts with a layer that defines the basic equation of state and then adds more layers of modeling that derive properties from previous layers. The *thermoType* entry in our case reads:

```
thermoType    hThermo<pureMixture<constTransport<specieThermo<hConstThermo<perfectGas>>>>>
```

where the following entries are

Thermophysical model: *hThermo*	General thermophysical model calculation based on enthalpy h.
Mixture properties : *pureMixture*	General thermophysical model calculation for passive gas mixture.
Transport coefficient: *constTransport*	Constant transport properties.
Derived thermophysical properties: *specieThermo*	Thermophysical properties of species, derived from c_p, h.
Basic thermophysical properies: *hConstThermo*	Constant specific heat c_p with avaluation of enthalpy h.
Equation of state: *perfectGas*	Perfect gas equation of state.

The basic themopysical properties are specified for each species from input data. The data is specified using a compound entry with the following format for a specie accessed through the keyword *mixture* :

```
mixture          Ar 1 39.948 520 0 3.4079e-05 0.65;
```

where

- *mixture*- keyword;
- *Ar*- name of specie;
- *"1"*- number of molecules of specie n_{moles};
- *"39.948"*- molecule weight $W(kg/kmol)$;
- *"520"*- heat capacity at constant pressure c_p $(J/(kmolK))$;
- *"0"*- ΔH $(J/kmol)$;
- *"3.4079e-05"*-dynamic viscosity (kg/ms);
- *"0.65"*-Prandtl number $Pr = \mu c_p/\kappa$, where κ is a thermal conductivity (W/mK).;

The first three parameters are used in *specieThermoI.H*, the heat capacity and ΔH are used in *hconstThermoI.H*, and the last two values are presented in $constTranspotI.H$

Turbulent properties

The *buoyantFoam* solver can be used for laminar flow as well as for turbulent flow. The *turbulentProperies* dictionary is set in *const/turbulentProperies*. For more details see Chapter 8.2 User Guide. In this tutorial we are solving our problem for the laminar case. For that we set *"turbulenceModel laminar"* and *"turbulence off"*.

Solution

Before the case is run, we will set the control data and schemes. The schemes that we will use to solve our problem were set in *control/fvSchemes* dictionary. Input data relating to the control of time reading and writing of the solution data are read from the *control/controlDict* dictionary. First we set the *startTime=0* and *endTime=2*. The *deltaT* represents the time step that should satisfy the CFL condition (Courant number $Co = \frac{\delta t|U|}{\delta x} \leq 1$). For more details see Chapter 2.1 User Guide.

To run our case type

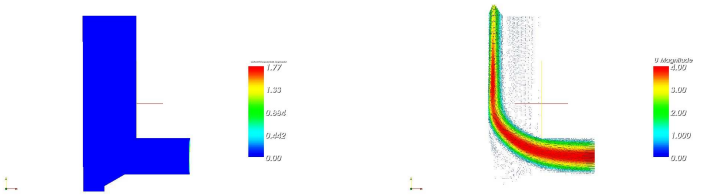
```
buoyantFoam . buoyantFoamTIG
```

in a command prompt. As soon as the results are written to time diretory, they can be viewed using *paraFoam*. For more details on how to use paraFoam see further Chapter 2.1.4 in the User Guide.

To view the results type

```
paraFoam . buoyantFoamTIG
```

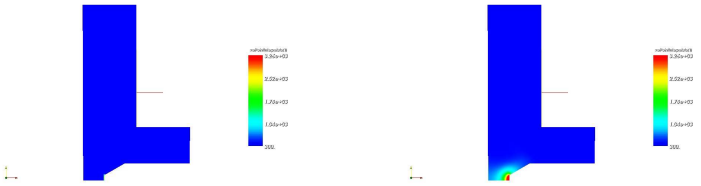
The velocity profiles at time T=0 and T=0.64



(a) Velocity profile at T=0. (b) Velocity profile at T=0.64.

Figure 4: The velocity profiles at time T=0 and T=0.64.

The temperature distribution at time T=0 and T=0.64



(a) Temperature distribution at T=0. (b) Temperature distribution at T=0.64.

Figure 5: The temperature distribution at time T=0 and T=0.64

Create new thermophysical model

Sometimes one needs to implement new models due to the *OpenFOAM* model/solver limitations. In our case, the simulation of the real fluid part in the TIG welding process would deal with higher temperatures, than those that one has in the standard tutorial case. That is, we cannot use the constant transport properties.

In this case we should create our own thermophysical model. The simplest way to create the new model is to copy the already existing one to the user directory (for example, copy to OpenFOAM/username-1.4/srs/thermophysicalModels) and modify it accordingly to ones needs. Then change the make files and options. You should also specify the path to user folder. For example, for *specie* we should modify the last line in *Make/files*:

```
LIB =\$(FOAM_USER_LIBBIN)/libspecieTIG
```

Then we should type *./Allwmake* or one can use the command *libso* to compile the new libraries and create the links. To make sure that you succesfully compiled new the libraries, go to the *OpenFOAM/username-1.4/lib/linux64Gcc4DPOpt*. For our case, one should be able to see the file *libspecieTIG.so*. This file is the link to the new library. After creating the new library, one should also make sure that the solver is actually using it.