OpenFOAM Pre-Processing in GME

Motivation

My research is the computational modeling of air-breathing engines. In the past year I have built and tested sub-sonic ramjet engines, as well as developed several simple two dimensional models of fluid flow. However, there is a need to compare the experimental data gathered from my ramjet testing with simulated data. For this use, basic 2D flow simulations will not suffice. There is a need for rigorous three dimensional simulations of not only fluid flow, but also combustion, spray atomization, and evaporation. For this purpose, I have turned to OpenFOAM, an open source library of CFD solvers.

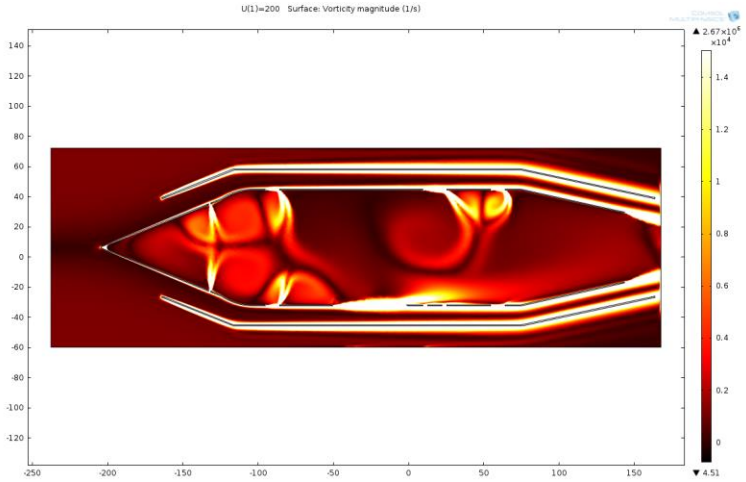

Figure 1 A Two Dimensional Ramjet Model

As will be explained in more detail in the Background section, OpenFOAM is a complex beast. In order to properly model and run simulations for my research, I will use GME to aid in setting up OpenFOAM "cases", the file directory that OpenFOAM reads to perform a simulation.

Background

Computational Fluid Dynamics (CFD) is broken into 3 distinct steps: 1) Pre-Processing 2) Solving and 3) Post-Processing. Pre-Processing is where the domain and scope of the simulation is defined, initial conditions set, and boundary conditions demarcated. Solving is generally using numerical techniques to solve a system of differential equations and define a set of state variables. Post-Processing is when the state variables solved in the previous step are analyzed and graphically displayed. Each of these three steps requires a vastly different set of skills to execute.

OpenFOAM is a collection of about 250 applications built upon over 100 software libraries. Each application performs a specific task with a CFD workflow. Examples of a utility within OpenFOAM is the pre-processing utility *snappyHexMesh* that is used to generate a mesh (a group of nodes and elements) around an arbitrarily defined geometry. OpenFOAM is great at handling the last two steps of the CFD process, Solving and Post-Processing. Where it is lacking is in the Pre-Processing step. While pre-processing tools are available to the user, the vast amount of possibilities for modeling and simulation of

an arbitrary object and physics make it impossible for OpenFOAM to provide any kind of automation to the process.

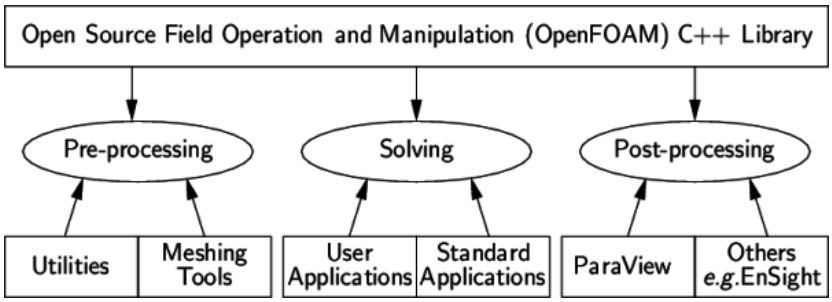

Figure 2 OpenFOAM Design

OpenFOAM cases are defined using a "case" folder. Each case folder contains three sub-folders: A "system" folder containing solve-specific settings, a "constant" folder where non-state variable constants and geometries are defined, and a "0" folder corresponding to time=0 where initial conditions are set.

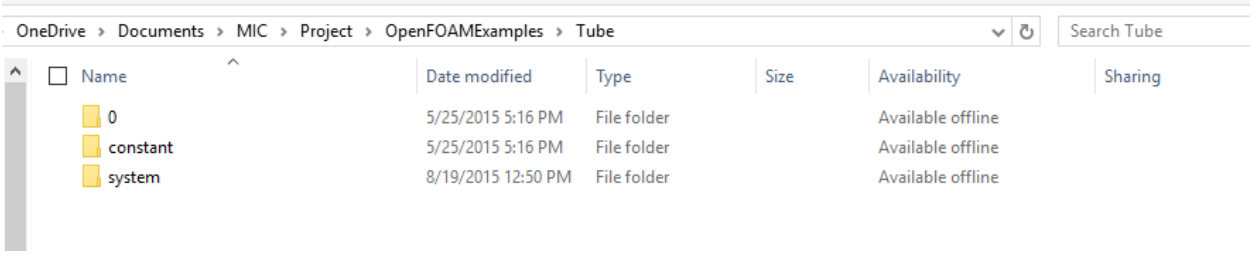

Figure 3 OpenFOAM Case Structure

Project Goals

The goal of this project is to develop a model of the OpenFOAM pre-processing step in GME. In particular, I will construct a model that is relevant to analysis of air-breathing engines. The model must be based upon the OpenFOAM case structure. Furthermore, an interpreter will be written to automatically convert the model into a valid and ready to use OpenFOAM case structure.

Meta-Model Approach

The meta-model will be split into two logical parts: 1) A set of common, reusable components that will be used to create 2) a solver-specific model.

Common components include properties such as Temperature, Velocity, and Pressure and settings such as numerical schemes, thermophysical models, and case settings. Solve specific models will fully describe a single OpenFOAM solver using the pre-made common components. As an example, this project contains a single fully defined solve, *rhoSimpleFoam*.

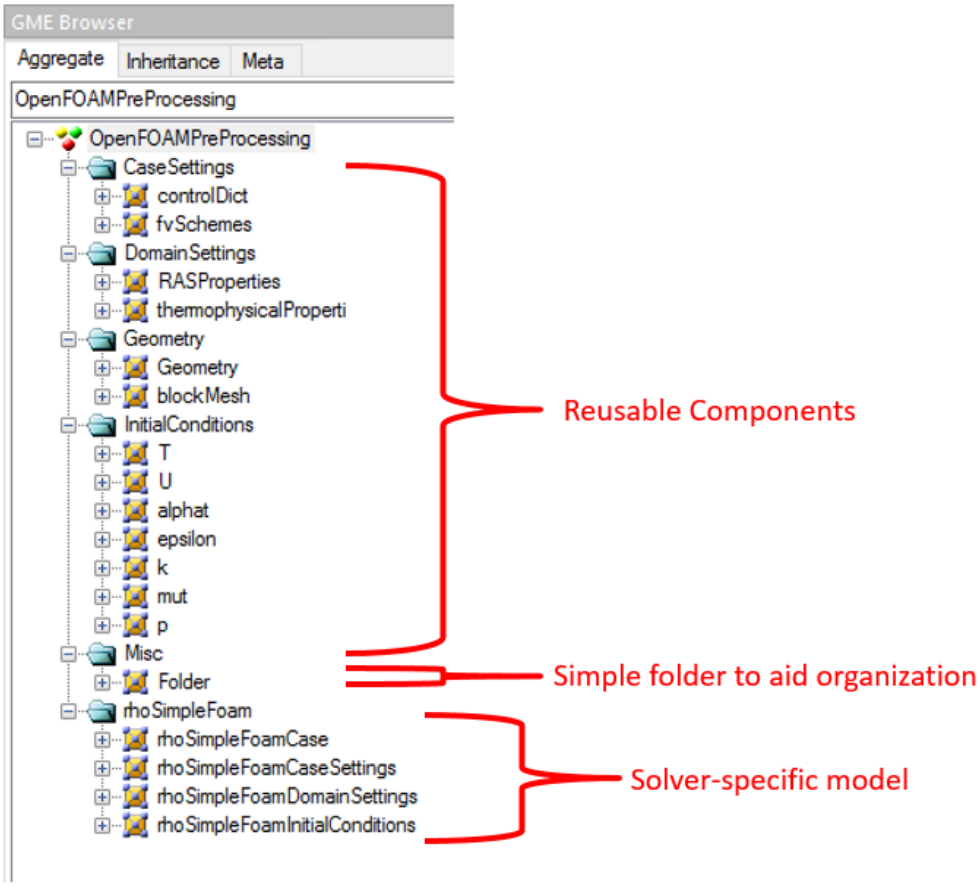

Figure 4 Meta-Model Hierarchy

rhoSimpleFoam is a solver that iteratively solves a system of differential equations for seven state variables: Velocity (U), Temperature (T), density(p), turbulent energy (k), turbulent viscosity (epsilon), and dynamic viscosities (mut,alphat). Therefore, *rhoSimpleFoamInitialConditions* will contain the properties model for each of these.

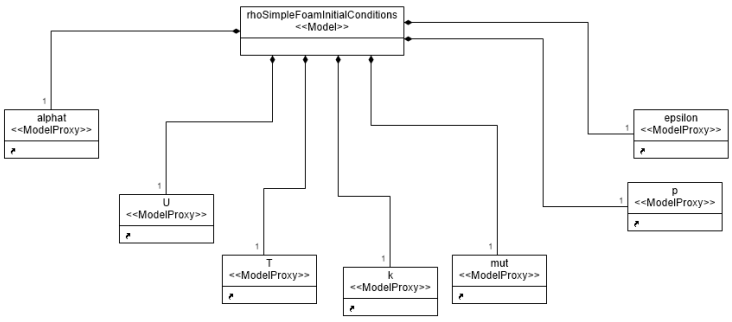

Figure 5 rhoSimpleFoam State Variables Defined in rhoSimpleFoamInitialConditions

Model Detail

During the model step, a solver is instantiated and its properties defined. Cardinality rules insure that every needed property set in the meta-model is defined within the model.

The user is required to define boundary conditions for each property based on the geometry being used. Each face, or patch, of a geometry object has to be constrained in some way.

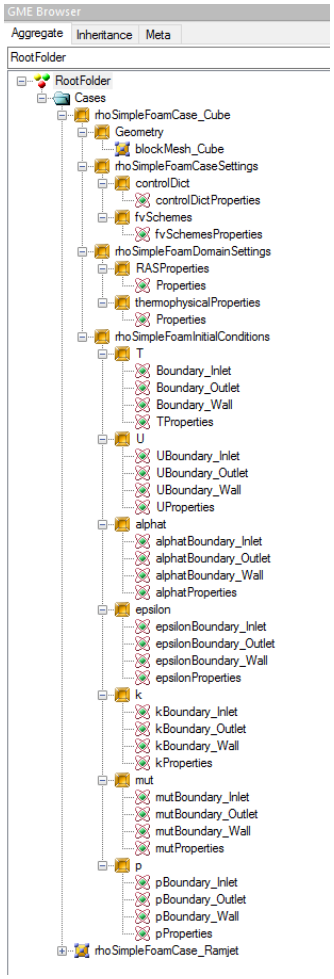

Figure 6 A fully defined rhoSimpleFoam Case

Interpreter

The interpreter was written in python due to its ease of use and the large amount of file generation and processing within the project. In order to process a GME model in python, an intermediary step must be taken. The GME model is first exported to a uml file, which is then read into a universal data model object within the python script. This UDM object can then be processed and iterated over to create the OpenFOAM cases defined within the model.

Writing in python allows the Cheetah templating engine to be used, which vastly reduced the lines of code needed to be written. Precompiled templates of each file that could be present within a case structure were made. When a property is encountered within the UDM data object, the corresponding template is instantiated and the file is created. This process means that all the python interpreter is concerned with is finding all relevant data nodes within the UDM object and passing it to the correct cheetah template. See below for a comparison of a boundary definition within a file and its corresponding template snippet.

```
boundaryField
{
    front
    {
        type            zeroGradient;
    }
    back
    {
        type            zeroGradient;
    }
    wall1
    {
        type            zeroGradient;
    }
    wall2
    {
        type            zeroGradient;
    }

    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           $internalField;
    }
}
```

```
boundaryField
{
#for $child in $node.children()
#if 'Boundary' in $child.type.name
    $child.patchName
    {
        type            $child.btype;
#if 'inletOutlet' in $child.btype
        inletValue      $child.inletValue;
#end if
#if 'compressible::turbulentMixingLengthDissipationRateInlet' in $child.btype
        mixingLength    $child.mixingLength;
#end if
#if 'turbulentIntensityKineticEnergyInlet' in $child.btype
        intensity       $child.intensity;
#end if
#if not ('zero' in '$child.btype')
        value           $child.value;
#end if
    }
#end if
#end for
}
```