NTNU – Trondheim
Norwegian University of
Science and Technology

A Graphical User Interface for the Computational Fluid Dynamics Software OpenFOAM

Henrik Kaald Melbø

Chemical Engineering and Biotechnology
Submission date: June 2014
Supervisor: Heinz A. Preisig, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Abstract

A graphical user interface for the computational fluid dynamics software OpenFOAM has been constructed. OpenFOAM is a open source and powerful numerical software, but has much to be wanted in the field of user friendliness. In this thesis the basic operation of OpenFOAM will be introduced and the thesis will emerge in a graphical user interface written in PyQt. The graphical user interface will make the use of OpenFOAM simpler, and hopefully make this powerful tool more available for the general public.

Sammendrag

Et grafisk grensesnitt for fluid dynamikk programmet OpenFOAM har blitt laget. OpenFOAM er en gratis og kraftig nummerisk programvare, men mangler endel når det kommer til brukervennlighet. Denne oppgaven beskriver den grunnleggende bruken av OpenFOAM og munner ut i et grafisk grennsesnitt skrevt i PyQt. Det grafiske grensesnitet vil gjøre det bruke OpenFOAM enklere og forhpendligvis, gjøre dette kraftfulle verktøyet mer åpent for allmenheten.

Preface

This thesis is the culmination of a five years master program at NTNU. It entails a journey on a crocked road known as discovery. Before picking up OpenFOAM, I had never touched on the subject of computational fluid dynamics. Yes, we had learned about fluids, Bernoulli's equations, and even had the pleasure of gazing at the Navier-Stokes equations and make vague guesses at its meaning. But we had not gained a deeper understanding of the concept that is "Computational Fluid Mechanics".

To pick up a new subject, a new software and a whole new understanding of the world, is not easily done. It requires time and patiences, and is greatly mediated by guidance. I must therefore thank my professor, Heinz A. Preisig, for guiding me down this road. Many of our discussions led to other subjects than what is encompassed in this thesis, and I am glad to say I picked up more this term than just knowledge of computational fluid dynamics.

Declaration of compliance

I declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).

Place and date: Trondheim May 4, 2014 Signature: Henrik Kaald Melbø

Contents

List of Figures

Chapter 1

Introduction

Computational fluid dynamics (hereby denoted as CFD) and model based solving has become significantly more important in the recent years, largely due to the growing availability of computational power. Computational fluid dynamic solutions generally require the repetitive manipulation of thousands, or even millions, of variables per case, this is humanly impossible without the aid of a computer.
Computer models gives the wielder an inexpensive, comprehensive and powerful tool to shape and test their system, before having to build expensive test plants. This will give an indication of feasibility and create a "sandbox" environment in which the creator may test, improve and solidify the end product.
For example, after the coming of the supercomputer, the calculation of the aerodynamic characteristics of new airplane designs via application of CFD has become more economically beneficial than measuring the same characteristics in a wind tunnel. In addition to economics, CFD offers the opportunity to obtain detailed flow-field information, some of which is difficult to measure in a wind tunnel [1].
The software OpenFOAM, offers a wide variety of solvers for CFD problems. The fact that it is an open source software makes it a tempting candidate when choosing a software to work with. The first major problem in using this software is to figure out how to operate it. OpenFOAM is not as user friendly as one may hope. To deal with this problem, the author has taken it upon himself to create a graphical user interface (GUI) for OpenFOAM, to help open up the world of free CFD calculations to the general public.

Chapter 2

Theory

2.1 Computational Fluid Dynamics

"A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join it and flow with it." -Frank Herbert

The subject, computational fluid dynamics, is a merging of several engineering subjects such as fluid mechanics, numerical mathematics and computer science. CFD is particularly dedicated to fluids that are in motion, and how the fluid-flow behaviour influences processes that may include heat transfer and possibly chemical reactions in combustion flows.
From ancient aqueducts to the modern airplain, fluids in motions has always occupied the human mind. Ancient civilisations used a crude form of fluid mechanics when creating projects for flood protection, irrigation, drainage, and water supply [2]. The first attempt to have a deeper understanding for the phenomena was done by Archimedes, when he in his book "On Floating Bodies" lay the foundation of hydrostatics [3]. The field of fluid mechanics has been in constant expansion since then, gaining additions from great minds such as Pascal, Newton, Bernoulli, d'Alembert, Euler, Dubuat, Helmholtz, Navier, Stokes and Reynolds.
The underlying concept of CFD used today, is based on continuum mechanics and its equations, meaning the substance is thought of as being made up of a continuous substance and not discrete particles. The conservation of mass and energy are underlying assumptions of this behaviour [4].

2.2 Navier-Stokes Equations

The foundation of modern fluid mechanics is the so called Navier-Stokes equations. These equations named after Claude-Louis Navier and George Gabriel Stokes, makes it possible to predict the velocity and pressure fields of a fluid in motion.

The Navier-Stokes equations is based on Newtons 2. law of motion. As formally stated in Principia: "The alteration of motion is ever proportional to the motive force impressed ; and is made in direction of the right line in which that force is impressed." [5]. This gives rise to the 2. Law of motion, also known as the conservation of momentum (equation 2.1).

$$\sum \boldsymbol{F} = m\boldsymbol{a} \tag{2.1}$$

Which states that force equals mass times the acceleration.
Using an Eulerian frame of reference, shown in figure 2.1 [4], equation 2.1 can be rewritten as equation 2.2.

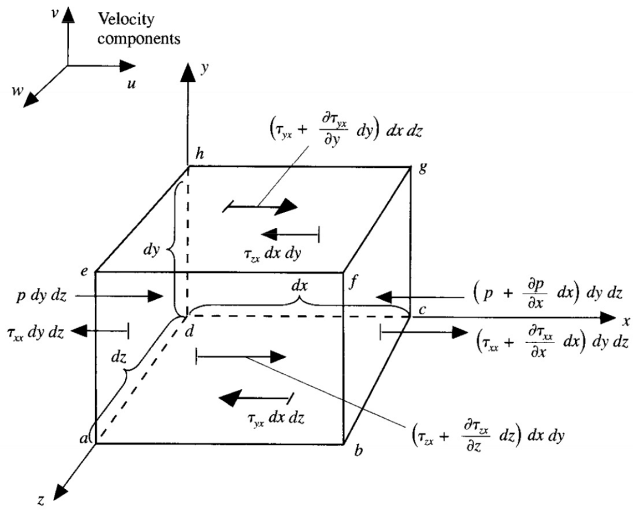

Figure 2.1: Acting Forces on a Control Volume

$$\sum \boldsymbol{F} = \frac{D\left(m\boldsymbol{v}\right)}{Dt} + \frac{\partial\left(m\boldsymbol{v}\right)}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial\left(m\boldsymbol{v}\right)}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial\left(m\boldsymbol{v}\right)}{\partial z}\frac{\partial z}{\partial t} \tag{2.2}$$

Where D is the substantial derivative, t is time, and $\boldsymbol{v}$ is the velocity.
Now momentum change per unit volume must be accounted for to get the convection. The working forces are: gravity, pressure and viscosity, other forces may also be present, depending on the case [4]. In general, the Navier-Stokes equations are the sum of these force, as given in equation 2.3.

$$\boldsymbol{F}_{pres} + \boldsymbol{F}_{grav} + \boldsymbol{F}_{visc} + \boldsymbol{F}_{misc} = m\boldsymbol{a} \tag{2.3}$$

For a infinitesimal volume ($dxdydz$) with uniform density, the convection terms of the momentum change as a function of time are given in equation 2.4.

$$\rho\left[\frac{\partial\left(|\boldsymbol{v}|\right)}{\partial t} + \frac{\partial\left(\boldsymbol{v}\right)}{\partial x}\frac{\partial x}{\partial t}\cdot\boldsymbol{i} + \frac{\partial\left(\boldsymbol{v}\right)}{\partial y}\frac{\partial y}{\partial t}\cdot\boldsymbol{j} + \frac{\partial\left(\boldsymbol{v}\right)}{\partial z}\frac{\partial z}{\partial t}\cdot\boldsymbol{k}\right] \tag{2.4}$$

Where $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$ is a unit vector for direction.
The change in x, y and z are the velocity terms as given in equation 2.5.

$$\begin{aligned} \frac{\partial x}{\partial t} &= \boldsymbol{v_x} \\ \frac{\partial y}{\partial t} &= \boldsymbol{v_y} \\ \frac{\partial z}{\partial t} &= \boldsymbol{v_z} \end{aligned} \tag{2.5}$$

And equation 2.4 can be rewritten as equation 2.6.

$$\rho \left[\frac{\partial \left(|\boldsymbol{v}| \right)}{\partial t} + \frac{\partial \left(\boldsymbol{v} \right)}{\partial x} \boldsymbol{v_x} \cdot \boldsymbol{i} + \frac{\partial \left(\boldsymbol{v} \right)}{\partial y} \boldsymbol{v_y} \cdot \boldsymbol{j} + \frac{\partial \left(\boldsymbol{v} \right)}{\partial z} \boldsymbol{v_z} \cdot \boldsymbol{k} \right] \tag{2.6}$$

The gravitational forces working on the element are given by equation 2.7.

$$\boldsymbol{F}_g = \rho \boldsymbol{g} dxdydz \tag{2.7}$$

Where ρ is the density and g is the gravitational constant.
The forces from pressure (surface stress acting normal and inward) acting on the element are given by equation 2.8.

$$\boldsymbol{F}_p = -\nabla p \cdot dxdydz \tag{2.8}$$

Where p is pressure and $\nabla = \frac{\partial}{\partial x} \cdot \boldsymbol{i} + \frac{\partial}{\partial y} \cdot \boldsymbol{j} + \frac{\partial}{\partial z} \cdot \boldsymbol{k}$.
The forces from viscosity acting on the element are given by equation 2.9.

$$\boldsymbol{F}_p = -\nabla \tau \cdot dxdydz \tag{2.9}$$

Shear stress (τ) has three couples per direction as shown in equation 2.10.

$$\boldsymbol{F}_p = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} \tag{2.10}$$

The shear stress is proportional to shear stress rate, as illustrated in equation 2.11.

$$\tau_{xy} = \tau_{yx} = \mu \left(\frac{\partial v_y}{\partial x} + \frac{\partial v_x}{\partial y} \right) \tag{2.11}$$

Here μ is the dynamic viscosity. Giving equation 2.12.

$$\tau_{xx} = -\frac{2}{3} \mu \nabla \cdot \boldsymbol{v} + 2\mu \frac{\partial v_x}{\partial x} \tag{2.12}$$

And similar for y, and z direction.

Putting this all together gives the general form of the Navier-Stokes equations (equations 2.13 - 2.15).

$$\rho g_x + \frac{\partial p}{\partial x} + \mu \left[\frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_x}{\partial y^2} + \frac{\partial^2 v_x}{\partial z^2} \right] = \rho \left[\frac{\partial \boldsymbol{v}_x}{\partial t} + \frac{\partial \boldsymbol{v}_x}{\partial x} \boldsymbol{v_x} + \frac{\partial \boldsymbol{v}_x}{\partial y} \boldsymbol{v_y} + \frac{\partial \boldsymbol{v}_x}{\partial z} \boldsymbol{v_z} \right] \tag{2.13}$$

$$\rho g_y + \frac{\partial p}{\partial y} + \mu \left[\frac{\partial^2 v_y}{\partial x^2} + \frac{\partial^2 v_y}{\partial y^2} + \frac{\partial^2 v_y}{\partial z^2} \right] = \rho \left[\frac{\partial \boldsymbol{v}_y}{\partial t} + \frac{\partial \boldsymbol{v}_y}{\partial x} \boldsymbol{v_x} + \frac{\partial \boldsymbol{v}_y}{\partial y} \boldsymbol{v_y} + \frac{\partial \boldsymbol{v}_y}{\partial z} \boldsymbol{v_z} \right] \tag{2.14}$$

$$\rho g_z + \frac{\partial p}{\partial z} + \mu \left[\frac{\partial^2 v_z}{\partial x^2} + \frac{\partial^2 v_z}{\partial y^2} + \frac{\partial^2 v_z}{\partial z^2} \right] = \rho \left[\frac{\partial \boldsymbol{v}_z}{\partial t} + \frac{\partial \boldsymbol{v}_z}{\partial x} \boldsymbol{v_x} + \frac{\partial \boldsymbol{v}_z}{\partial y} \boldsymbol{v_y} + \frac{\partial \boldsymbol{v}_z}{\partial z} \boldsymbol{v_z} \right] \tag{2.15}$$

The Navier-Stokes equations are notoriously difficult to handle, no known analytical solution to the problem is known. The Navier-Stokes equations has made it to the famous "Millennium Problems" list of mathematical problems issued by the Clay Mathematics Institute (CMI), along with other famous problems such as the "Riemann Hypothesis" and the "P vs. NP Problem" [6].

2.2.1 Solving the Navier-Stokes Equations

In the absence of a analytical solution to the Navier-Stokes equations, numerical schemes are used to solve a discretized version of the equations.
In essence, discretization is the process of which a function (which are viewed as having infinite continuum of values throughout some domain) is approximated by analogous expressions which prescribe values at only a finite number of discrete points or volumes in the domain [4].

Commonly used discretization schemes are the finite difference method (FDM), finite volume method (FVM) and finite element method (FEM).

FDM approximates the Navier-Stokes equations as discrete points in space. It is the easiest to implement, but have a difficulties along curved boundaries and has some general mesh adaptation problems [7].

FVM approximates the Navier-Stokes equations as a system of (cell-wise) conservation equations. This has the advantage of being based on physical conservation properties, but this method has some problems on unstructured meshes [7].

FEM approximates the Navier-Stokes equations in their variational form with high order polynomial trial functions. This has the advantage of being highly accurate, but has problems when dealing with complex domains [7].

2.2.2 Algorithms for Solving the Discretized Navier-Stokes Equations

There are many approaches to solving the discretized Navier-Stokes equations. Some well known algorithms for a incompressible case, are the ICE (Implicit Continuous-Fluid Eulerian) method and the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) method [8].

The ICE method is a semi-implicit method, let's for example consider an inviscid iso-thermal flow. From the momentum equation there is possible to derive an Helmoltz equation for the pressure p^{n+1} as seen in equation 2.16.

$$-\frac{p^{n+1}}{\alpha^n (\Delta t)^2} + \nabla^2 p^{n+1} = -\frac{p^n}{\alpha^n (\Delta t)^2} + \frac{1}{\Delta t}\nabla \cdot [p\boldsymbol{u}^n - \nabla \cdot (\nabla \cdot (p\boldsymbol{uu})^n)] \quad (2.16)$$

Where, α is a linearisation constant, arising from linearisation of pressure $p^{n+1} = \alpha^n (p - p^n)$. With this, the density and velocity at t^{n+1} can be obtain [8].
Many adaptation of these algorithms has been made, such as the Stability-Enhancing Two-Step (SETS) method [9], Fully implicit ICE (FICE) method [10], and others.

The SIMPLE method is an implicit method, which introduces a cycle of successive corrections for the velocity and the pressure terms until convergence [8]. Decoupling the momentum and continuity equations and solving the pressure and velocity fields independently.
The SIMPLER (SIMPLE Revised) [11] develops this further to include temperature coupling. A non iterative version of SIMPLE procedure, called PISO (Pressure implicit with splitting of operator), uses fractional steps [12]. The PISO algorithm is similar to SIMPLE, but applies no under-relaxation and the corrector step for momentum is done more than once.

2.3 Preconditioners

The numerical solving of the linear system $\boldsymbol{Ax} = \boldsymbol{b}$ is the most time consuming process of any CFD computation. The term "preconditioning" refers to transforming this system into another system with more favourable properties for iterative solution. Generally speaking, preconditioning attempts to improve the spectral properties of the coefficient matrix [13]. The system will be in the form $\boldsymbol{M^{-1}Ax} = \boldsymbol{M^{-1}b}$, where $\boldsymbol{M}$ is the preconditioner.

Some matrices are easier to solve than others, meaning they require less computations, such as lower triangular and upper triangular matrices. LU (Lower Upper) decomposition, factors a matrix as the product of a lower triangular matrix and an upper triangular matrix, $\boldsymbol{A} = \boldsymbol{LU}$. Thus simplifying the system.

Incomplete-Cholesky is a sparse approximation of the Cholesky factorization, where decomposition of a positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose (similar to LU decomposition). When used with a conjugate gradient method, very remarkable convergence accelerations on symmetric, positive definite matrices, have been obtained, as shown by Kershaw [14]. The coupling with the conjugate gradient method is essential, since the matrix obtained from the incomplete-Cholesky decomposition, although close to unity, still has a few extreme eigenvalues, which will be dominant at large iteration numbers [15].

2.4 Meshes

When the equations are discretized, the geometry also requires subdivision of the domain into a number of smaller, non-overlapping sub-domains in order to solve the flow physics, the network of these sub-domains are frequently called a grid or a mesh. There exist many different mesh geometries such as tetrahedral and hexagonal structures. It is important to have tight meshes at edges and in regions it is expected to be high activity since the granularity of the mesh directly affect the resolution of the calculated field [16].
A mesh consists of nodes, edges, faces and (in case of 3D-problems) volumes. A node is the equivalent of a vertex in geometry, and it is these points that is calculated at each step in a solver. An example of a cubic volume is shown in figure 2.2.

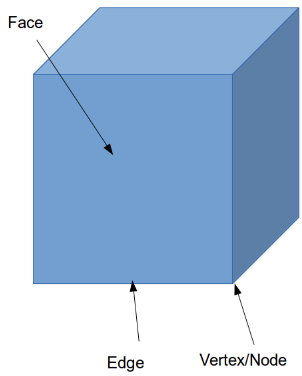

Figure 2.2: Illustration of a Cubical Volume in a Mesh

Figure 2.3 shows an example of a tetrahedral mesh of a torus generated by the Netgen algorithm in Salome. This mesh has 26050 nodes, 315 edges, 299976 faces and 106785 volumes. The faces makes up volumes, called cells. A cell is a list of faces in arbitrary order. Cells must have the following properties:

- Contiguous
- Convex
- Closed
- Orthogonality

Contiguous means the cells must completely cover the computational domain and must not overlap one another. Every cell must also obviously be mathematically convex and topologically closed. In addition the face-area vector (the dot product of the area vector of a face) and and the centre-to-centre vector (a vector from the centroid of the cell to the centroid of that face) must always be less than 90

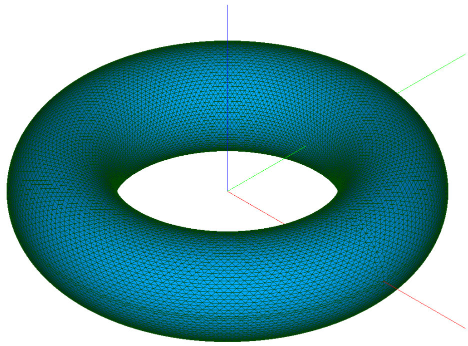

Figure 2.3: Example of a Mesh Generated via the NETGEN Algorithm on a Torus

degrees. The orthogonality angel greatly affects the mesh quality [17].

Many different cell and grid types are available. Choice of grid depends on the problem and the solver capabilities. Some examples of cell types are given in fig 2.4 [18]. There are also hybrid meshes (combination of different cells), multiblock

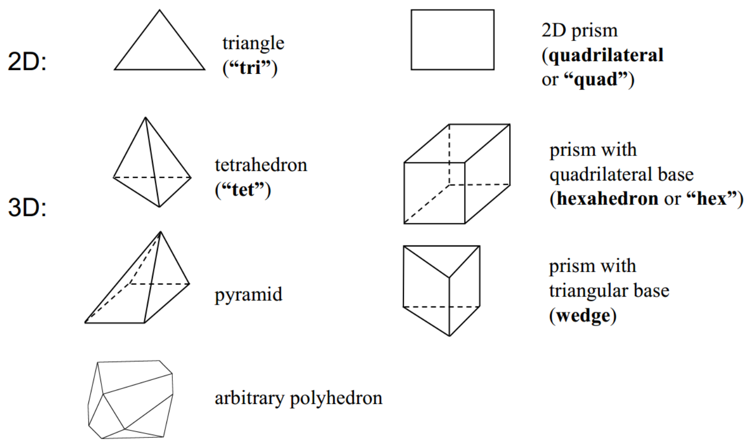

Figure 2.4: Example of Cell/Element Types

(different meshes ”blocked” together) and nonconformal meshes (in which grid nodes do not match up along an interface).
In some cases, it is difficult to ensure adequate grid resolution, when necessarily flow features are unknown. A feature known as solution-based grid adaption deals with this problem [18]. The grid will then be refined or coarsened by the solver based on the developing flow.

OpenFOAM, by default handles all meshes as a ”polyMesh”. A polyMesh is de-

fined as a mesh of arbitrary polyhedral cells in 3-D, bounded by arbitrary polygonal faces i.e. the cells can have an unlimited number of faces where, for each face, there is no limit on the number of edges nor any restriction on its alignment. This offers greater freedom in mesh generation, but can also lead to some problems converting imported meshes. OpenFOAM offers several "cellShape" tools to help with such problems.
The polyMesh description is based around faces, and internal faces connect 2 cells and boundary faces address a cell and a boundary patch. Each face is assigned an owner cell and neighbour cell so that the connectivity across a given face can simply be described by the owner and neighbour cell labels. In the case of boundaries, the connected cell is the owner and the neighbour is assigned the label -1. So in order to describe the entire mesh, the following information needs to be given:

- points, a list of vectors describing the cell vertices
- faces, a list of faces, each face being a list of indices to vertices in the points list
- owner, a list of owner cell labels, the index of entry relating directly to the index of the face
- neighbour, a list of neighbour cell labels
- boundary, a list of patches

Patches are described by a dictionary, giving information of start face and number of faces affected. An example:

```
movingWall
{
    type patch;
    nFaces 20;
    startFace 760;
}
```

2.5 Computer Aided Drawing

To create a geometrical model of a system, so that a mesh can be constructed, a mathematical description of the system is needed. This can be done by hand, describing each grid point and defining the boundaries by giving them coordinates in the Cartesian system. But as the system grows large and complicated, this is no longer a trivial task. Describing curves and complex patters using coordinates alone becomes increasingly difficult as the system grows, especially if working with an irregular mesh (where the spacing of the grid is non-uniform). To illustrate this principle, the geometry of a very simple case (given as a tutorial in OpenFOAM as Lid-driven cavity flow) is illustrated in figure 2.5. This simple design, consisting of a box, needs the following specifications to be complete as a mesh for OpenFOAM.

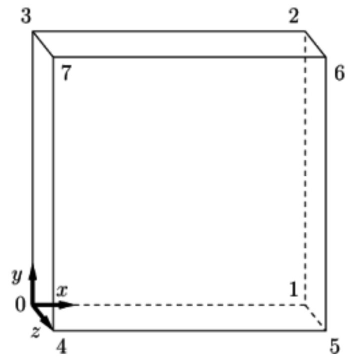

Figure 2.5: Lid-driven Cavity Flow Geometry

```
convertToMeters 0.1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
```

```
                (0 4 7 3)
                (2 6 5 1)
                (1 5 4 0)
            );
        }
        frontAndBack
        {
            type empty;
            faces
            (
                (0 3 2 1)
                (4 5 6 7)
            );
        }
);

mergePatchPairs
(
);
```

Now try to extrapolate this example to something more complex, such as an airplane wing. To aid in this endeavour, a computer aided drawing (CAD) software is a helpful tool. This will let you create surprisingly complex models of (almost) any shapes in 2-D or 3-D space, using a wide variety of tools. It should be noted that OpenFOAM handles all meshes as a "polyMesh", as can be constructed from this code by running the "blockMesh" command in the terminal.

An example of such a CAD software is Salome. Salome is open source and enables the user to make complex geometrical objects and also has the capability to generate meshes on these objects. Creating a mesh can be quite computationally heavy, especially if there is many volumes involved. Salome has a default limit of 500 000 elements, but this can easily be changed for those cases needing a tighter mesh. Any grid over 1 000 000 elements proved laborious and when pushing up towards 10 000 000 elements, problems allocating memory occurred, this is of course computer dependant, and given a powerful enough computer, extremely tight meshes can be made.

When constructing a mesh, many factors must be considered in regards to shape and optimization. Salome offers a variety of mesh generating algorithms such as NETGEN, to assist in the construction. NETGEN generates triangular or quadrilateral meshes in 2-D, and tetrahedral meshes in 3-D. The input for 2-D is described by spline curves, and the input for 3-D problems can be defined by Constructive Solid Geometry. The algorithm offers automated topology based mesh size control (with user controllable constraints) and mesh refinement algorithms [19].

2.6 OpenFOAM

When the geometry is accounted for, the equations must be solved. Since solving the Navier-Stokes equations numerically by hand is out of the questions, some pow-

erful tools are required. There are plenty of good software candidates for the task on the market, such as Fluent, Abaqus, Star-CD, FLOW-3D, CFX and others. The program used in this thesis is the Open Source Field Operation and Manipulation (OpenFOAM), which is a open source software package for CFD written in C++ specially designed to solve field equations. The package contains solvers, a meshing tools (Salome, NX or other CAD programs can also be used), post-processing tools (paraFoam, an adaptation of paraView) and, being open source, has room for code customization. This will allow the user to add their own equations and/or algorithms. OpenFOAM solves any kind of continuum fields, such as pressure, velocity, temperature and even electric fields.
The beauty of OpenFOAM lays not in its simplicity, but in its adaptivity. Being an open-source software, OpenFOAM is continually growing and improving by a worldwide collaborative effort of CFD enthusiasts. Because everyone has access to the code, anyone has the opportunity to improve on its contents. The user can "peel" off the exterior of the software and inspect the "machinery" that lays beneath, giving more understanding and customization opportunities. Drawbacks to this is of course the possibility of unknown bugs, and the fact that no one is obliged to help solve any problems or answer any questions that may appear.

2.6.1 Creating a Case in OpenFOAM

Starting out with OpenFOAM can be quite confusing, there are many files and folders which needs to be at the right place for a case to be run properly. Most often, an unspecific error message will return after trying to execute the program. This section tries to frame a generic case, and guide the user through the setup process.

Firstly, the geometry must be accounted for. In OpenFOAM, meshes can be directly implemented (manually entered) or be made in an external program (such as Salome) and then imported and converted to the polyMesh standard. In Salome, the mesh can be saved in I-Deas (.unv) format and then put in the OpenFOAM case directory. The command "ideasUnvToFoam" run in this folder will then transform the file to a polyMesh in the appropriate folder. It is necessary that general case structure (especially the "system" folder) is already present, with the structure shown in figure 2.6, as OpenFOAM reads in these dictionaries before creating the mesh.

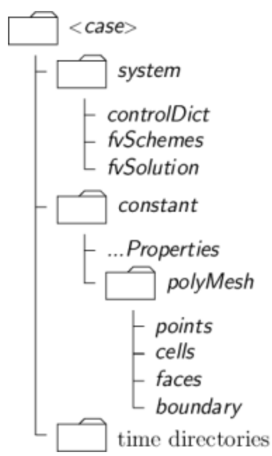

Figure 2.6: Structure of a OpenFOAM Case

The following actions and choices will then have to be manually entered in various C++ documents in the case folder (see figure 2.6).

system: Contains the information on the solvers used and other information relevant to read and write operations of the program. The systems folder needs 3 dictionaries, the "controlDict", "fvSchemes" and "fvSolutions".

controlDict: All OpenFOAM solvers begin all runs by setting up a database. The controlDict dictionary sets input parameters essential for the creation of this database. Here time control and data writing parameters are specified.

fvScheme: The fvSchemes dictionary sets the numerical schemes for terms, such as derivatives, gradients, interpolation etc., in equations, that appear in applications being run. OpenFOAM offers complete freedom to choose from a wide selection of schemes. The terms that can be specified are:

- *interpolationSchemes*: Point-to-point interpolations of values.
- *snGradSchemes*: Component of gradient normal to a cell face.
- *gradSchemes*: Gradient calculations ∇.
- *divSchemes*: Divergence calculations $\nabla \bullet$.
- *laplacianSchemes*: Laplacian calculations. The Gauss scheme is the only choice of discretisation, ∇^2.
- *timeScheme*: First and second order time derivatives $\frac{\partial}{\partial t}$, $\frac{\partial^2}{\partial^2 t}$.
- *fluxRequired*: Fields which require the generation of a flux.

fvSolutions: Contains information the equation solvers, tolerances and algorithms used. Necessary specifications are:

- *solver*: The linear algebra solver controller, such as Preconditioned Conjugate Gradient Method (PCG), Preconditioned Bi-Conjugate Gradient Method (PBiCG) and Generalised Geometric-Algebraic Multi-Grid (GAMG) etc.
- *tolerance*, relTol or maxIter: The sparse matrix solvers are iterative, i.e. they are based on reducing the equation residual over a succession of solutions.
- *preconditioner*: Options for preconditioning of matrices in the conjugate gradient solvers.
- *smoother*: The solvers that uses a smoother, such as GAMG, require the smoother to be specified.

"constant": This folder contains the polyMesh data files and the necessary physical properties (such as turbulence models, transport properties etc.) needed for the given application.

As the running applications writes data to the system, time directories will be created. The initial conditions at starting time step, must be added manually as a folder (”0” (for $t = 0$) if startTime is set to zero in the controlDict). In this folder the boundary conditions are specified for the needed properties must be specified. The boundaries will be the same as have been specified on the mesh, any undefined faces of the mesh will be named defaultFaces (if non was specified, all faces of the mesh will be one boundary named defaultFaces). There are several options for boundary conditions, the syntax for defining them is the same in all cases, here is an example for a boundary named ”defaultFaces”:

```
defaultFaces
{
type            <type>;
value           <value>;
}
```

The type can be given as following:

- fixedValue; a list of values (often from previous calculated simulations).
- fixedGradient; the normal gradient is specified.
- zeroGradient; normal gradient is zero.
- calculated; Mixed fixedValue/ fixedGradient.

The value is then given as either a vector (x y z), as for velocity, or as a scalar when dealing with pressure.
There is also more advanced derived properties such as ”pressureInletVelocity”, for instruction on use of these options, see the manual or go to http://www.openfoam.org/docs/user/boundaries.php.

When all this is in place (for a first case, try copying the tutorial folders and modify them to the intended case) the case can be run with the appropriate terminal command, ”simpleFoam” for the simpleFoam solver, ”icoFoam” for the icoFoam solver etc., from the case folder.

As the user has probably discovered by now, the software requires quite a learning effort. Knowledge of the appropriate syntax, the location and requirements of different dictionaries in different folders for different solvers according to different boundary conditions. All which is subject to change for different solvers. This is of course only a matter of time and experience on the usage of the software, but the overall impression is quite chaotic. A useful extension of this powerful tool would be a simple graphical user interface to guide the user.

2.7 Parallel Computing

A feature that have revolutionized CFD, is so called parallel computing. Parallel computing will allow the user to address different processes to different cores in

a multi-core processor computer. This will decrease the solving time for larger problems.
The method of parallel computing is already implemented by OpenFOAM and is known as domain decomposition. In domain decomposition the geometry and associated fields are broken into pieces and allocated to separate processors for solution. The mesh and fields are decomposed using the ”decomposePar” utility. The underlying aim is to break up the domain with minimal effort but in such a way to guarantee a fairly economic solution. OpenFOAM offers several ways of splitting domain.

- Simple geometric decomposition in which the domain is split into pieces by direction, e.g. 2 pieces in the x-direction, 1 in y etc.
- Hierarchical geometric decomposition which is the same as simple except the user specifies the order in which the directional split is done, e.g. first in the y-direction, then the x-direction etc.
- Scotch decomposition which requires no geometric input from the user and attempts to minimise the number of processor boundaries.
- Manual decomposition, where the user directly specifies the allocation of each cell to a particular processor.

A decomposed OpenFOAM case is run in parallel using the openMPI. Message Passing Interface (MPI) is a language-independent communications protocol used to program parallel computers.

In OpenFOAM the syntax will be:
mpirun −−hostfile <machines> -np <nProcs> <foamExec> <otherArgs> -parallel > log &
Where: <nProcs> is the number of processors; <foamExec> is the executable, e.g. icoFoam; and, the output is redirected to a file named log.

According to Amhdahl’s law the speed-up can be calculated with the following formula $S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{B+\frac{1}{n}(1-B}$. Where $T(1)$ is the runtime with 1 processor, $T(n)$ is the runtime with n processors and B is the fraction of the algorithm which is strictly serial. Amdahl’s law gives a relationship between the expected speed-up of the parallelized implementations of an algorithm relative to the same algorithm in series [20].

2.8 Graphical User Interface

It is not necessary to go back more than 30 years before most computer interactions was mostly text based. To communicate with the computer, the user needed to enter commands into a text based command line. So when the Macintosh was introduced in 1984, it represented something altogether new to the public; an affordable Graphical User Interface.

A GUI is what most people today think about when the word ”computer program” is mentioned. The magical window which by the help of a easily navigated pointer, lets you manipulate by dragging and dropping icons, the intricate workings of a computer.
Nowadays, deeper knowledge of the ”innards” of a computer, the low level assembly language, and even the high level computer languages are hidden, and in parts forgotten, behind the shining mask of a GUI. Even if this sounds like quite a limiting agent, the fact is that the human mind is more adept at solving graphical problems than purely logical statements.
Using assembly language, the seemingly simple procedure of adding 2 numbers is transformed to the daunting task of understanding the following statement:

```
.model small
.data
 opr1 dw 1234h
 opr2 dw 0002h
 result dw 01 dup(?),'$'
.code
        mov ax,@data
        mov ds,ax
        mov ax,opr1
        mov bx,opr2
        clc
        add ax,bx
        mov di,offset result
        mov [di], ax

        mov ah,09h
        mov dx,offset result
        int 21h

        mov ah,4ch
        int 21h
        end
```

Higher level languages simplify this for the user, allowing for the simple algebraic statements, but the code is ultimately reduced to assembly when it is compiled.
So, why don’t we use high-level languages to do computations? Well, we do. Almost all modern programs is written in some form of high-level language such as C, C++, JAVA, Pyhon etc. Assembly is mostly useful when writing compilers (often not even then) for these higher level languages. But for the general public, even these pieces of code are difficult to comprehend and a bit intimidating. A GUI therefore acts as a bridge between the communities, giving anyone free access to the use of the program, no prior knowledge of computer languages, algorithms or even the workings of the program itself is needed.
This lays a heavy burden on the shoulders of the developer, as he is the users guide in the usage of the program. The user may not know any presumptions or liberties the program may take, and will often trust blindly in the output.

2.9 PyQt

To create a GUI, the right tools are needed. Qt is a cross-platform application framework that is widely used for developing application software, such as GUI. Qt uses standard C++. An adaptation of the Qt framework, called PyQt, provides binding between Python and Qt, allowing for the same frame work as Qt, only in written in Python. PyQt is developed by the British firm Riverbank Computing and is available as a GNU General Public License [21].
One may ask: "Why choose PyQt and not Qt. Isn't C++ faster?". The simple answer is that I am more accustom to Python, and prefers the language over C++. C++ as a compiled language is certainly faster than Python (being an interpreted language) in all aspects, but since Qt bindings for Python uses C++ compiled Qt anyway, there is actually no difference in runtime. One should also mention that the runtime of the GUI is practically nothing compared to the runtime of the underlying CFD software.

In order to make the source code, found in the appendix somewhat readable, a short introduction to PyQt will be given. For more thorough explanation the reader is refereed to "Rapid GUI Programming with Python and Qt" [22].

2.9.1 PyQt Basics

Most applications has one "Main Windows" with several "Dialogs" or "Widgets". Widgets are interfaces used for tasks and communications with the user. Dialogs are the top-level window (often pop up), mostly used for short-term tasks and brief communications with the user. This may include check-boxes, radio buttons, text input, and other features such as "Accept" and "Reject" buttons.

A PyQt application may be seen as an event system. Signals are what changes the systems state.

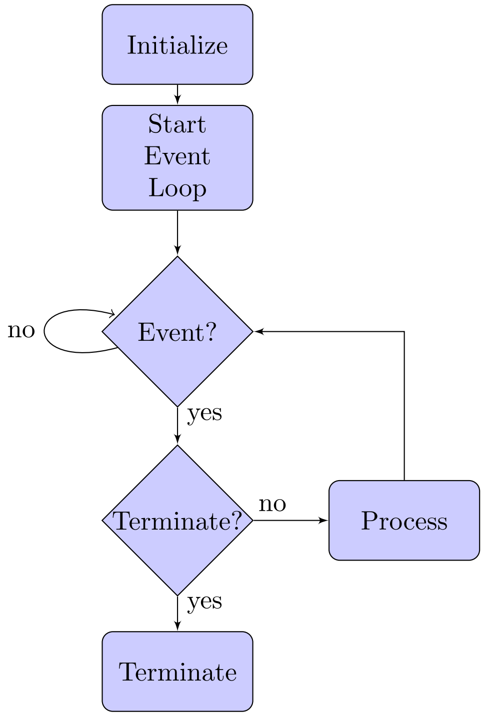

Figure 2.7: Structure of a GUI Application

As seen can be seen in figure 2.7, the application will keep running in an infinite loop, until a signal triggers an event. In PyQt, the signal can take the form:

```
self.pushButton.clicked.connect(self.selectFile)
```

This small piece of code, makes a button named ”pushButton” run the function called ”selectFile” when it is clicked. The ”self.” simply refers to the class it is incorporated in (in most cases, this is the ”Main” function, as signals often are declared in the primary loop).
There are many types of signals, some triggers when a combobox changes value, some may trigger when a chackbox is checked/unchecked, and some may even trigger on other signals, creating a chain of events. The signal/slot mechanism has the following features:

- A signal may be connected to many slots.
- A signal may also be connected to another signal.
- Signal arguments may be any Python type.
- A slot may be connected to many signals.
- Connections may be direct (ie. synchronous) or queued (ie. asynchronous)

- Connections may be made across threads.
- Signals may be disconnected.

Where "slot" is a Python callable (an action).

PyQt supports "rich text", meaning that HTML and CSS markup can be directly implemented into the PyQt code. The user can for example incorporate the following line in a PyQt document:

```
self.textEdit_2.setToolTip(_translate("openFOAM", "<html><head/><body><p><span style=\"font-family:\'arial,sans-serif\'; font-size:12px; color:#000000; background-color:#ffffff;\">Solver tolerance</span></p></body></html>", None))
```

To customise the input areas. This may look messy, but gives the user a powerful tool when manipulating graphics. Note how there is plain HTML markup in this line of code. Colour is defined by hexadecimal, and font-size is set as normally is done in HTML.

When creating dialogs, the user can either write pure code and modify every aspect by setting the initial properties, or use a program, such as Qt Designer. Qt Designer is a visual design tool for creating Qt and PyQt dialogs. In Qt Designer, the user can simply drag and drop elements and compose the basic layout for the GUI, even signals can be implemented. A combination of pure code and Qt Designer can also be used, using the visual tool to create the layout, and importing it into another script, adding another layer of functionality on top of the design.

To demonstrate a simple PyQt program, a simple "Hello, World!" application is shown below.

```
"""
A simple "Hello, World!" PyQt program
Demonstrating buttons, signals and slots
"""
# To get access to OS (print in terminal),
# python needs to import the "sys" module
import sys

# This is the PyQt library
from PyQt4 import Qt

# Instantiate a QApplication by passing
# the arguments of the script to it:
a = Qt.QApplication(sys.argv)

# Our function/slot to call when the button is clicked
def sayHello():
    print "Hello, World!"

# Instantiate the button
hellobutton = Qt.QPushButton("Say 'Hello world!'",None)

```

```
# And connect the action "sayHello" to
# the event "button has been clicked"
a.connect(hellobutton, Qt.SIGNAL("clicked()"), sayHello)

# Set a modal dialog and returns control to the caller
hellobutton.show()

# This executes the loop
a.exec_()
```

This code launches a button. This button is connected via a signal to the slot "sayHello" which prints out "Hello, World!" on the terminal, when pressed.

Chapter 3

Results

3.1 A GUI for OpenFOAM

OpenFOAM has many extremely good qualities, user-friendliness can not exactly be characterised as one of them. The user will have to navigate around in folders and interact with several different C++ documents, adding commands and input in various locations to give input to the C++ executable program, in addition the whole process is operated from the command terminal, adding another layer of confusion to the already dazzled user. This might not frighten the old-school hardy FORTRAN programmer who in his youth solved everything using only punch-cards and a trusty IBM 709. But this will probably frighten the more inexperienced users away from this powerful, and not to mention free, software.
An effort to build a guided user interface would be a extremely helpful tool to lower the barrier, and allow a broader user group of the software.

3.1.1 Development

The OpenFOAM framework encompasses many options and has much room for customization. When creating the GUI several factors are important:

- Simplicity
- Consistent behaviour
- Fast learning curve
- Permutable actions
- Feedback
- Default actions

Simplicity implies that the options should be obvious and self-explanatory, dialogues and actions should be easy to utilize. The behaviour of the GUI should be consistent and lead to expected behaviour which guides, and not confuses the user. The grasping of the software shpuld be intuitional. The options and actions should be permutable, meaning they should be reversible and the user should have the option to easily alter previous action. The GUI should also give feedback and guide the user in making the right choices. The default values will be discussed in length in the next section.

3.1.2 Default Values

Setting up a GUI gives the creator quite a lot of power. What the user sees as "Default" set values, has an impact on the usage of the software, often the default values will not be changed, and may not even be considered by the user. The default values should therefore be set with some forethought, as to be helpful and not to mislead the user. Some problems where encountered when setting up the "fvSchemes" and "fvSolutions" options in the GUI. How much of the information should we let the user set? All of it? In that case the amount of options and text in the GUI became unmanageably and ugly. But the user needs to be able to set most, if not all of these options. The answer became to give the user one single button for each option. This button simply opened up the file for either "fvSchemes" or "fvSolutions" in a new window, allowing manual editing of solvers and schemes. This was not as elegantly implemented as was wished for, but gave the user much more unrestricted access to the mechanics of the program.

3.2 OpenFOAM GUI Documentation

The usage if the GUI will now be demonstrated. The code for the GUI can be found in Appendix A-E. Appendix A covers the "Main Program". Here, all PyQt elements are initialized. Appendix B covers the "Interface", meaning the visual layout of the GUI generated in Qt Designer. The ".UI" file, generated from Qt Designer, is converted by the command "pyuic4 input.ui -o output.py" to usable Python code. This code is a bit messy and incorporates some HTML and CSS, but can easily be incorporated in the main program, by importing the script and setting up the UI with the inbuilt Qt function "setupUi()". Appendix C covers the "Utility Functions", which are classes which are called by the main program to do different useful tasks. Appendix D covers the solver modules and Appendix E, the turbulence modules.

It should be noted that this software was developed on a UBUNTU operating system, with OpenFOAM version 2.2.2. The code is written in PyQt 4, and has not been tested outside these conditions.

When launching the GUI (run "./openFOAM.py" from terminal in main folder), the first interface that appears, gives the user a choice of solvers, a short description of the solver chosen and the choice of creating a new case, or loading an older case. This is shown in figure 3.1. In addition, the option of launching paraFOAM will be given when a case is either loaded or created. If the user tries to load a mesh or run additional commands without first creating/loading a case, a helpful error message will appear.

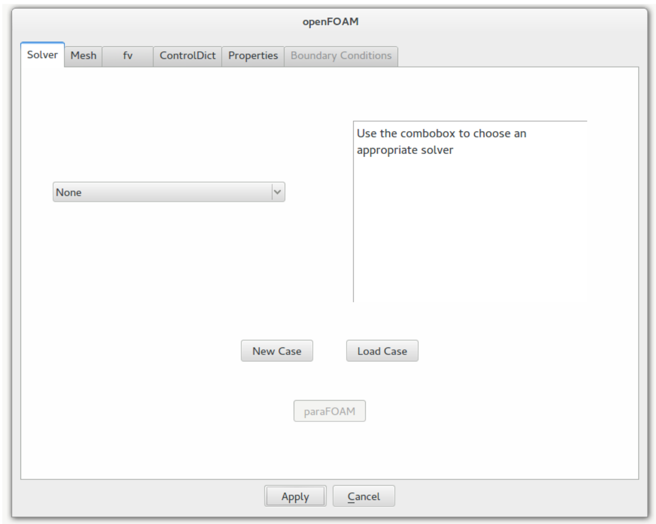

Figure 3.1: Solver Tab of GUI

As can be seen in figure 3.1, the tabbed interface gives the opportunity to navigate the different options and necessary inputs which is needed to run the case. Not all options will be available before some information is added, for example; no boundaries may be found until a mesh is loaded.
After creating/loading a case, the user may navigate to the "Mesh" tab (Figure 3.2) and import a mesh.

Any mesh format that OpenFOAM allows, such as Fluent (.msh), I-DEAS (.UNV) and GAMBIT (.NEU), are implemented, or (if necessary) the user may manually add a mesh into the case folder.

The "fv" tab (figure 3.3) allows the user to open a windows which allows manual editing of mathematical schemes and solver options. Originally these were implemented as a series of text-inputs and comboboxes, but this proved to restraining and messy. A more elegant, but less "in the style of the GUI philosophy" approach was therefore implemented. This gives the user more control (alas also more room for error).

To set the control variables, the user navigates to the "ControlDict" tab (Figure 3.4). Here the user specifies a start time (in case some previous data is stored in the casefile), the time length the simulation is to run, the "timestep" (Δt) of the solver and how often the data should be saved to file. To have continuous fields,

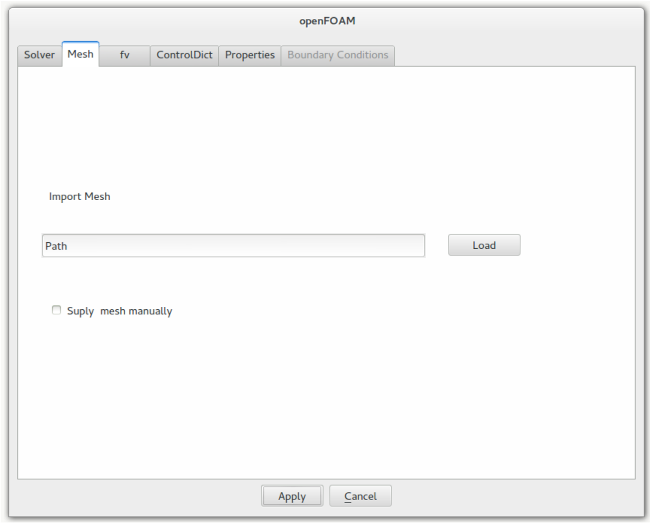

Figure 3.2: Mesh Tab of GUI

a low "Write interval" is necessary, but be mindful that this will slow down the runtime (as writing to ROM is slow vs. writing to RAM), and in addition take more space.

After having specified the control variables, the user navigates to the "Properties" tab (Figure 3.5). Here he may initiate turbulence (which spawns a brand new tab) and change the viscosity for the system. There is also added an emulated terminal command line if additional commands should be needed. Here the user may run useful commands such as "checkMesh" (which checks validity of a mesh), "renumberMesh" (which renumbers the cell list in order to reduce the bandwidth), or miscellaneous post-processing operation and other useful OpenFOAM utilities.

If a previous case with a mesh is loaded in, or a new mesh is added, the GUI automatically finds and adds the boundary conditions to the "Boundary Conditions" tab (Figure 3.6). This tab will now be accessible and the user may specify the boundaries. The boundaries need a "Type and a "Value" as described in the theory section. To help the user a autocomplete feature is built-in.

In case of tubulence, an extra tab will be added to the GUI, named "Turbulence Properties" (Figure 3.7). This will allow for setting the boundary conditions for the turbulent flow.
When all necessary choices are made, the simulation can commence. The "Accept"

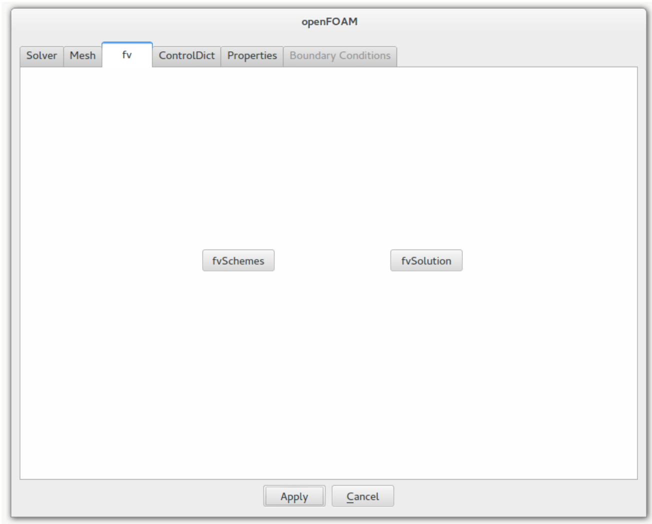

Figure 3.3: Fv Tab of GUI

button will apply the chosen options and launch the solver. If there are any inconsistencies in the input, a helpful error message will pop up, guiding the user to the faulty or missing section. If the simulation clears the preliminary check, a loadbar will launch, signifying a successfully launch. At the end of the simulation, a new window displaying data or errors (if the simulation crashed) will appear. The data can be saved, and an option for launching paraFOAM is also given.

3.3 PyQt Structure

The structure of the PyQt program has been subject to alteration multiple times, starting out as an unmanageable lump of continuous code, and culminating in a modular structure as shown in figure 3.8.

The openFOAM.py file contains the main body of the program. The graphical part of the interface is located in the ”intr” (Interface) folder, ”outp” is the temporary storage of all outputted (from terminal) data, ”pix” contains graphics, the ”root” folder contains all necessary templates for OpenFOAM cases, ”solv” contains easily manipulated modules of solvers (see next section), ”turb” contains similar modules for turbulence, and ”util” contains the necessary utility functions called in the openFOAM.py script.

3.3.1 Solvers and Turbulence

Since OpenFOAM is quite easy to modify and add custom solvers and utilities, the GUI gives room for expansions. By adding a simple script with the necessary

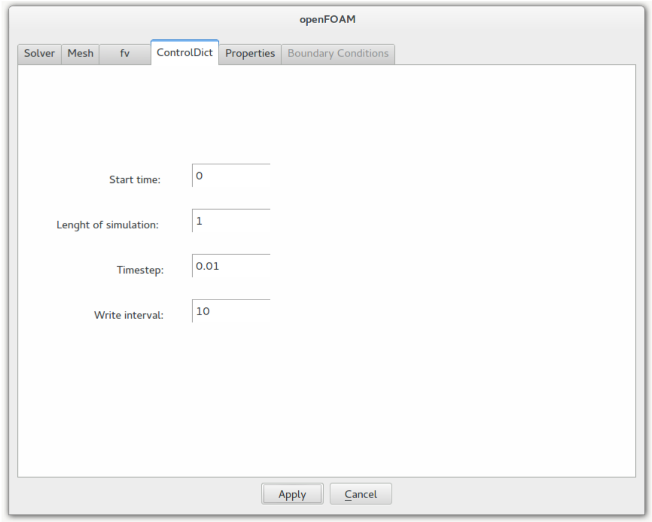

Figure 3.4: ControlDict Tab of GUI

information needed on either the solver or the turbulence model in the appropriate folder, the GUI incorporates this into it's framework at next run of the program. Here is shown the code for the "simpleFOAM" solver in the solv folder.

```
import solver

def simpleFoam():
    s = solver.Solver("simpleFoam",
                      "SimpleFoam is a steady-state solver for incompressible, turbulent flow",
                      {"U", "p"})
    return s
```

This simple file contains the necessary information the GUI needs to handle the solver; The callable name of the solver: "simpleFoam", a short description for the users benefit, and a list of the needed variables (and therefore templates) needed to set up the case. This script rolls this information nicely into an object and passes the information to the main script, incorporating it into the GUI. A similar setup is used for the turbulence models.

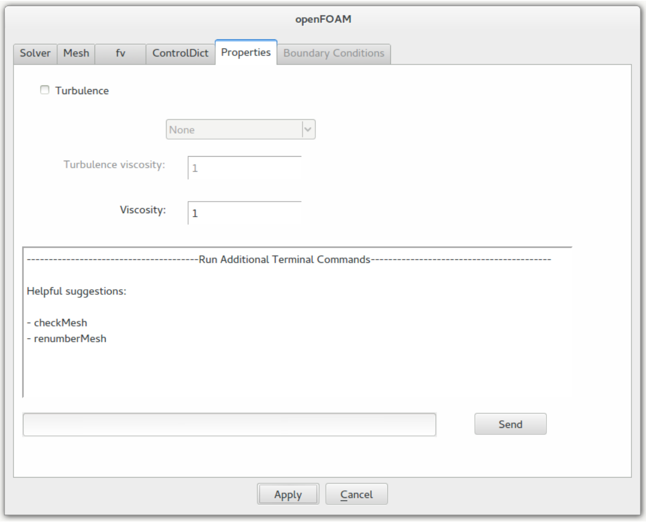

Figure 3.5: Properties Tab of GUI

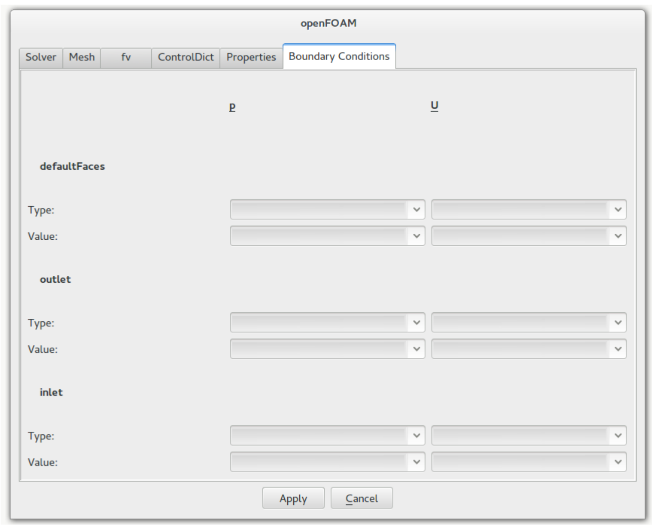

Figure 3.6: Boundary Condition Tab of GUI

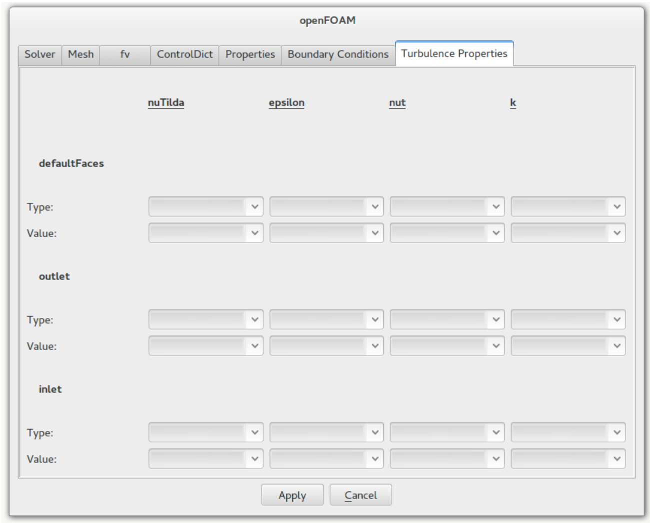

Figure 3.7: Turbulence Tab of GUI

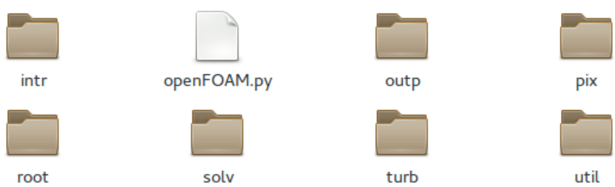

Figure 3.8: PyQt Program Folder Structure

Chapter 4

Discussion and Conclusion

4.1 Discussion

In creating the GUI, many choices as to the usage, and freedom of the user had to be made. Trying to encompass all the features of OpenFOAM in a GUI would be almost impossible, or atleast far outside the scope of this thesis. Many functions and utilities are custom made for specific solvers and situations and requires exceptions from regular operations. The goal in writing the GUI was therefore more to find a general solving method for problems. Taking away options, and not adding them. This may sound limiting for the user, but anyone who have tried OpenFOAM can testify that there is quite a barrier to overcome to use it, even for menial tasks. Hopefully the GUI will lower the bar for using this powerful and free software, and make it more open to the general public.

The hardest part in making this GUI, was not the programming, or the understanding of OpenFOAM, but rather what to include. In the beginning I tried to encompass everything. My belief was that all possible modulation and configurations which was possible in OpenFOAM should be included. The code swelled and bloated to impossible proportions, and more and more bugs and inconsistent behaviour followed. At that time it was even written all in one long script, impossible to edit and impossible to wrap your head around. Then I gained some sagely advice from my supervisor: "Redo it, start from scratch". I have to admit I was quite sceptical to that endeavour, thinking back on all my hard work getting this code to work, but it paid off.

Starting my butchers work in slicing and cutting away all superficial and unnecessary functions, pruning away all the dead weight and retaining a core of simple (or at least simpler), yet more functional code. The end result is only a fraction of what i started out with, but still, a much more usable and stable program.

4.2 Further Work

The GUI is operational, but far from complete. As OpenFOAM is forever expanding, the GUI also has the capability of incorporating new modules and adding new functionalities. As is shown in the previous section, solvers and turbulence models can easily be added. Even all of the basic OpenFOAM solvers are not implemented. This could easily be done, adding a script for every one of them. The GUI also has a lot to be wanted when it comes to the aesthetic. It looks quite gray and boring. Some experimentations with colors and icons where conducted, but did not improve the overall impression of the GUI. It was therefore concluded to leave it gray. Someone with a more artistic touch should probably be given the honours of decorating the GUI. There is probably also some ingenious features in OpenFOAM, not incorporated in the GUI. This should be remedied.

Bibliography

[1] Bruce R Munson. *Computational Fluid Dynamics, An Introduction*. Springer, Belgium, 2009.

[2] G. Garbrecht. *Hydrologic and hydraulic concepts in antiquity in Hydraulics and Hydraulic Research: A Historical Review*. A.A. Balkema, 1987.

[3] T. L. Heath. *The Works of Archimede*. Dover Publications, Inc, 2002.

[4] J. D. Anderson Jr. *Computational Fluid Dynamics, The Basics with Applications*. McGraw-Hill, 2009.

[5] I. Newton. *Principia, Mathematical Principles of Natural Philosophy*. Daniel Adee, 1846.

[6] Clay Mathematics Institute. Millennium problems, 2014. URL: http://www.claymath.org/millennium-problems, Accessed: 2014-05-31.

[7] R. Rannacher. Finite element methods for the incompressible navier-stokes equations. *Whitepaper*, 1999.

[8] H. Laval P. Pegon, A. Soria. A review of finite element solvers for the compressible navier-stokes equations. *Commission of the European Communities, Joint Research Center*, pages 34–38, 1993.

[9] J.H. Mahaffy. A stability-enhancing two-step method for fluid flow calculation. *Comput. Phys.*, 46:329–341, 1982.

[10] A. Latrobe. Fast finite difference methods for solving 1 d two-phase flow equations. *Paper presented at 3rd CSNI Specialists Meeting on Transient Two Phase Flow, Pasadena.*, 1982.

[11] S. V. Pantakar and D. B. Spalding. A calculation procedure for heat. mass and momentum transfer in threedimensional parabolic flows. *Int. J. Heat Mass Transfer*, 15:1787, 1972.

[12] R.I. Issa. Solution of the implicitly discretised fluid flow equations by operator-splitting. *J. Comput. Phys*, 62:40–65, 1986.

[13] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 2002.

[14] David S. Kershaw. The incomplete choleskyconjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26(1):43 – 65, 1978.

[15] C. Hirsch. *Numerical Computation of Internal and External Flows - Fundamentals of Computational Fluid Dynamics (2nd Edition)*. Elsevier, 2007.

[16] D. J. Mavriplis. Unstructured mesh related issues in computational fluid dynamics (cfd) based analysis and design. *Whitepaper*, 2002.

[17] OpenFOAM Foundation. OpenFOAM user guide, 2014. URL: http://www.openfoam.org/docs/user/, Accessed: 2014-05-31.

[18] A. Bakker. PyQt user guide, 2006. URL: http://www.riverbankcomputing.co.uk/software/pyqt/intro, Accessed: 2014-05-31.

[19] Joachim Schberl. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, 1997.

[20] Xian-He Sun and Yong Chen. Reevaluating amdahls law in the multicore era. *Journal of Parallel and Distributed Computing*, 70(2):183 – 188, 2010.

[21] Riverbank. Applied computational fluid dynamic, lecture 7 - meshing, 2014. URL: http://www.bakker.org/dartmouth06/engs150/07-mesh.pdf, Accessed: 2014-05-31.

[22] M. Summerfield. *Rapid GUI Programming with Python and Qt*. Prentice Hall, 2007.

Appendices

Appendix A: Main Program

```
#!/usr/bin/python
'''
GUI for openFOAM

 This program launches a guided user interface for openFOAM

 Updates:
 The solver are implemented as modules
 The general structure is changed

 Requirements: openFOAM 2.2.2 with paraFOAM running on UBUNTU
 Made by: Henrik Kaald Melb\o
NTNU, Trondheim, Norway
 Department Of Chemical Engineering
 Contact: henrikkaald@gmail.com
 Version: 3.0
 Date: 14. Mar. 2014
'''
from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt
from PyQt4.QtGui import *
from intr import UI
from util import TextEditer, Wiew, Loader, replace, textparse, fvSo
from solv import simpleFoam, icoFoam, pisoFoam, scalarTransportFoam
from turb import kEpsilon
import sys, os, time, subprocess, re, shutil
from tempfile import mkstemp
from shutil import move
from os import remove, close
import imp

##########################################################
#Initiating Main GUI                                     #
##########################################################
class Main(QDialog, UI.Ui_openFOAM):
    def __init__(self, parent = None):
        super(Main, self).__init__(parent)
        # Set up the user interface from Designer.
        self.setupUi(self)
```

```python
        # Connect up the buttons and comboboxes
        self.pushButton.clicked.connect(self.selectFile) #Load mesh button
        self.pushButton_3.clicked.connect(self.load) #Load case button
        self.pushButton_2.clicked.connect(self.new) #New case button
        self.pushButton_5.clicked.connect(self.fvSc) #Edit fvSchemes
        self.pushButton_7.clicked.connect(self.fvSo) #Edit fvSolution
        self.pushButton_6.clicked.connect(self.terminal) #Terminal command
        self.pushButton_4.setEnabled(False)
        self.textEdit.setEnabled(False)
        self.label_14.setEnabled(False)
        self.comboBox_4.setEnabled(False)
        self.pushButton_4.clicked.connect(self.para)
        self.connect(self.comboBox, QtCore.SIGNAL('activated(QString)'), self.combo) # Solvers
        self.connect(self.comboBox_4, QtCore.SIGNAL('activated(QString)'), self.turbComb) # Turbulence
        applyButton = self.buttonBox.button(QtGui.QDialogButtonBox.Apply)
        QtCore.QObject.connect(applyButton, QtCore.SIGNAL("clicked()"), self.accept)
        cancelButton = self.buttonBox.button(QtGui.QDialogButtonBox.Cancel)
        self.checkBox_2.stateChanged.connect(self.nomesh)
        self.textEdit_9.setReadOnly(True)
        self.textEdit_11.setReadOnly(True)
        self.checkBox.stateChanged.connect(self.turbulence) #Turbulence
        self.Schemes.setTabEnabled(6, 0)
        self.Schemes.removeTab(3)

        #Get solvers from solv folder
        cwd = os.getcwd()
        path = cwd+"/solv"
        files = os.listdir(path)
        solvers = set()
        for strings in files:
            pices = strings.rsplit('.',1)
            solvers.add(pices[0])
        solvers.discard("__init__")
        solvers.discard("solver")
        for solver in solvers:
            self.comboBox.addItem(solver)

    #Creates a new case folder
    def new(self):
        global case
        global pwd
        case = QFileDialog.getSaveFileName(self, "New case")
        case = str(case)
        if not case:
            del case
            return
        pwd = os.path.dirname(os.path.realpath(__file__))
        case = str(case)
        os.system("cp -r "+pwd+"/root "+case)
```

```
        self.pushButton_4.setEnabled(True)

    #Loads in an older case folder
    def load(self):
        global case
        global pwd
        global mesh
        case = QFileDialog.getExistingDirectory()
        if not case:
            del case
            return
        case = str(case)
        pwd = os.path.dirname(os.path.realpath(__file__))
        os.system("cp -r "+pwd+"/root/templates "+case)
        if os.path.isdir(case+"/constant/polyMesh"):
            self.findBound()
        self.pushButton_4.setEnabled(True)
        mesh = True

    #Get name of program to run, from combobox
    def combo(self, text): #Run
        if text == "None":
            self.textEdit_11.setText("No solver selected")
            try:
                s
            except:
                return
            del s
            return
        global program, properties
        program = str(text)
        exec "s = "+program+"."+program+"()"
        self.textEdit_11.setText(s.description)
        properties = s.properties
        #New solver can lead to a new set of boundaries
        try:
            mesh
        except:
            return
        self.findBound()

    #Turbulence Combobox
    def turbComb(self, text):
        if text == "None":
            self.Schemes.removeTab(7)
            try:
                s
            except:
                return
            del s
            return
        global turbprop
        program = str(text)
        exec "s = "+program+"."+program+"()"
        turbprop = s.properties
        self.initturbulence()

```

```
    #Initializing the final preparations for running the solver
    #Remaining checks and finializations
    def accept(self):
        #Read, search and replace parameters
        global delta
        global write
        try:
            case
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No case defined", QtGui.QMessageBox.Ok)
            return
        if not self.validate():
            return
        try:
            program
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No solver defined", QtGui.QMessageBox.Ok)
            return
        if program == "None":
            QtGui.QMessageBox.question(self, 'An error has occured', "No solver defined", QtGui.QMessageBox.Ok)
            return
        if not os.path.isdir(case+"/constant/polyMesh"):
            QtGui.QMessageBox.question(self, 'An error has occured', "No mesh defined", QtGui.QMessageBox.Ok)
            return
        if not self.condType():
            QtGui.QMessageBox.question(self, 'An error has occured', "Ill defined boundary type", QtGui.QMessageBox.Ok)
            return
        #Get controlDict
        self.control()
        #If turbulence
        try:
            self.turb
        except:
            self.turb = False
        if self.turb == True:
            self.setturbulence()
        #Write to viscosity
        self.visc()
        #Makes boundries reversable
        if not os.path.exists(case+"/"+str(start)):
            os.makedirs(case+"/"+str(start))
        for f in properties:
            shutil.copy2(case+"/templates/"+f+"_tmp", case+"/"+str(start)+"/"+f)
        #Check and set boundaries
        self.cond()
        #Set solver and run (program is run in the loader)
        load = Loader.Loader(case, run, delta, write, program, pwd, start)
        if load.Run() == 1:
            w = Wiew.Wiew(case, pwd)
            w.exec_()
```

```
    #Loads in mesh
    def selectFile(self): # Load and run mesh
        try:
            self.meshstate
        except:
            self.meshState = False
        if self.meshState == True:
            return
        global mesh
        global path
        try:
            case
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No case defined", QtGui.QMessageBox.Ok)
            return
        path = QFileDialog.getOpenFileName(self, "Load Mesh", case , "*.unv *.ans *.msh *.neu *.geo")
        if not path:
            self.lineEdit.setText("Mesh")
            del path
            return
        self.lineEdit.setText(path)
        path = str(path)
        QApplication.setOverrideCursor(QCursor(Qt.WaitCursor))
        #Copy boundarys
        suf = path.rsplit('.',1)
        os.system("cp " + path +" "+ case)
        mesh = path.rsplit('/',1)
        self.defBound(mesh, suf)
        QApplication.restoreOverrideCursor()
        self.findBound()
        #Create mesh, write info to "MehsInfo"

    #Changes the mesh state
    def nomesh(self, state):
        if state == QtCore.Qt.Checked:
            self.meshState = True

    #Allows for manual input in fvSchemes
    def fvSc(self):
        try:
            case
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No case defined", QtGui.QMessageBox.Ok)
            return
        T = TextEditer.TextEditer(case)
        T.exec_()

    def fvSo(self):
        try:
            case
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No case defined", QtGui.QMessageBox.Ok)
```

```
            return
        T = fvSo.fvSo(case)
        T.exec_()

    #Checks whats kind of mesh is supplied, and does the necessary conversion
    def defBound(self, mesh, suf):
        gr = open(pwd+"/outp/MeshInfo","w")
        if suf[-1] == "unv":
            d = subprocess.Popen(["ideasUnvToFoam", mesh[1]], cwd=case, stdout = gr, stderr = gr)
        elif suf[-1] == "neu":
            d = subprocess.Popen(["gambitToFoam", mesh[1]], cwd=case, stdout = gr, stderr = gr)
        elif suf[-1] == "msh":
            d = subprocess.Popen(["fluentMeshToFoam", mesh[1]], cwd=case, stdout = gr, stderr = gr)
        elif suf[-1] == "geo":
            d = subprocess.Popen(["cfx4ToFoam", mesh[1]], cwd=case, stdout = gr, stderr = gr)
        elif suf[-1] == "ans":
            d = subprocess.Popen(["ideasToFoam", mesh[1]], cwd=case, stdout = gr, stderr = gr)
        d.wait()
        gr.close()

    #Finds the boundaries from mesh
    def findBound(self):
        try:
            case
        except:
            return
        bound = case+"/constant/polyMesh/boundary"
        f = open(bound, 'r+')
        lines = f.readlines()
        f.close()
        num = int(lines[17])
        first = []
        for i in range(1, num+1):
            j = 13+6*i
            first.append(lines[j][:-1])
        #Create boundaries in U,P,T etc
        try:
            properties
        except:
            return
        for f in properties:
            shutil.copy2(case+"/templates/"+f, case+"/templates/"+f+"_tmp")
        global con
        con = len(first)
        for i in range(0, con):
            pb = first[i]+"\n    {\n        type            typ"+str(i)+"/*type*/;\n        value           val"+str(i)+"/*val*/; \n    }\n"
            for f in properties:
                textparse.parse(case+"/templates/"+f+"_tmp", pb, 23)
```

```
        #The next section creates the new "Boundaires" tab
        #Auto completer
        c = QtGui.QCompleter(["fixedValue", "fixedGradient", "mixed",
            "calculated", "directionMixed", "zeroGradient", "
            movingWallVelocity", "pressureInletVelocity", "
            pressureDirectedInletVelocity", "surfaceNormalFixedValue",
             "totalPressure", "turbulentInlet", "buoyantPressure", "
            inletOutlet", "outletInlet", "pressureInletOutletVelocity"
            , "pressureDirectedInletOutletVelocity", "
            pressureTransmissive", "supersonicFreeStream", "slip", "
            partialSlip", "patch", "symmetryPlane", "empty", "wedge",
            "cyclic", "wall", "processor"])
        c.setCompletionMode(QtGui.QCompleter.PopupCompletion)
        d = QtGui.QCompleter(['uniform (x y z)', 'uniform p'])
        d.setCompletionMode(QtGui.QCompleter.PopupCompletion)
        newtab = QtGui.QScrollArea()
        newtab.setWidget(QtGui.QWidget())
        newtab_layout = QtGui.QVBoxLayout(newtab.widget())
        newtab.setWidgetResizable(True)
        global edit
        edit = {}
        #grid positioner
        grid = QtGui.QGridLayout()
        font = QtGui.QFont()
        font.setUnderline(True)
        font.setBold(True)
        #Add appropriate lables
        n = 1
        for f in properties:
            lable = QtGui.QLabel(str(f))
            lable.setFont(font)
            grid.addWidget(lable,0,n)
            n +=1
        font.setUnderline(False)
        j = 1
        i = 1
        while len(first) > 0:
            label = QtGui.QLabel(first.pop())
            label.setFont(font)
            grid.addWidget(label, j, 0)
            label = QtGui.QLabel("Type: ")
            grid.addWidget(label, j+1, 0)
            label = QtGui.QLabel("Value: ")
            grid.addWidget(label, j+2, 0)
            for x in range(1, n):
                edit[(i)] = QtGui.QComboBox()
                grid.addWidget(edit[(i)], j+1, x)
                edit[(i+1)] = QtGui.QComboBox()
                grid.addWidget(edit[(i+1)], j+2, x)
                edit[(i)].setEditable(True)
                edit[(i+1)].setEditable(True)
                edit[(i)].setCompleter(c)
                edit[(i+1)].setCompleter(d)
                i += 2
            j += 3
        newtab.setLayout(grid)
        #Create a new tab, then switch
```

```
        self.Schemes.addTab(newtab, "Boundary Conditions")
        self.Schemes.removeTab(5)

    #Sets the control in ControlDict
    def control(self):
        global run, delta, write, start
        cd = case+"/system/controlDict"
        start = self.textEdit_12.toPlainText()
        start = str(start)
        run = self.textEdit_4.toPlainText()
        run = str(run)
        delta = self.textEdit_5.toPlainText()
        delta = str(delta)
        write = self.textEdit_6.toPlainText()
        write = str(write)
        shutil.copy2(case+"/templates/controlDict", case+"/system/controlDict")
        replace.replace(cd, "STAART", start)
        replace.replace(cd, "TIMEEND", run)
        replace.replace(cd, "TDELTA", delta)
        replace.replace(cd, "INTERVALWRITE", write)

    #Finds the boundary conditions (groups from geometry)
    def cond(self):
        i = 1
        for j in range(con-1, -1, -1):
            for f in properties:
                var = case+"/"+str(start)+"/"+str(f)
                typ = edit[(i)].currentText()
                val = edit[(i+1)].currentText()
                replace.replace(var, "typ"+str(j)+"/*type*/;", str(typ)+";")
                if typ == "zeroGradient":
                    replace.replace(var, "value           val"+str(j)+"/*val*/;", "")
                else:
                    replace.replace(var, "val"+str(j)+"/*val*/;", str(val)+";")
                i += 2

    #Checks if leagal boundary condition type is given
    def condType(self):
        text = set()
        i = 1
        for j in range(0, con):
            ty1 = edit[(i)].currentText()
            ty1 = str(ty1)
            text.add(ty1)
            i += 2
        values = set(["fixedValue", "fixedGradient", "zeroGradient", "calculated", "mixed", "directionMixed",
                    "movingWallVelocity", "pressureInletVelocity", "pressureDirectedInletVelocity",
                    "surfaceNormalFixedValue", "totalPressure", "turbulentInlet", "fluxCorrectedVelocity",
                    "buoyantPressure", "inletOutlet", "outletInlet", "pressureInletOutletVelocity",
```

```
                        "pressureDirectedInletOutletVelocity", "pressureTransmissive", "supersonicFreeStream",
                        "slip", "partialSlip", "patch", "empty", "symmetryPlane", "wedge", "cyclic",
                        "wall", "processor"])
            if text.issubset(values):
                    return True
            else:
                    return False

    #If turbulence is activated, get properties
    def turbulence(self, state):
        if state == QtCore.Qt.Unchecked:
            try:
                case
            except:
                self.textEdit.setEnabled(False)
                self.label_14.setEnabled(False)
                self.comboBox_4.setEnabled(False)
                self.Schemes.removeTab(7)
                self.turb = False
                return
            self.noturbulence()
            self.turb = False
            return
        try:
            case
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No case defined", QtGui.QMessageBox.Ok)
            self.checkBox.setCheckState(QtCore.Qt.Unchecked)
            return
        if state == QtCore.Qt.Checked:
            self.turb = True
            self.textEdit.setEnabled(True)
            self.label_14.setEnabled(True)
            self.comboBox_4.setEnabled(True)

    #Setup the turbulence tab
    def initturbulence(self):
        for f in turbprop:
            shutil.copy2(case+"/templates/"+f, case+"/templates/"+f+"_tmp")
        shutil.copy2(case+"/templates/turbulenceProperties", case+"/constant/turbulenceProperties")
        bound = case+"/constant/polyMesh/boundary"
        f = open(bound, 'r+')
        lines = f.readlines()
        f.close()
        num = int(lines[17])
        first = []
        for i in range(1, num+1):
            j = 13+6*i
            first.append(lines[j][:-1])
        global tcon
        tcon = len(first)
        for i in range(0, tcon):
```

```python
            pb = first[i]+"\n    {\n        type            typ"+str(i)+"/*type*/;\n        value           val"+str(i)+"/*val*/; \n    }\n"
            for f in turbprop:
                textparse.parse(case+"/templates/"+f+"_tmp", pb, 23)
        #Need to add options in tab
        newtab_2 = QtGui.QScrollArea()
        newtab_2.setWidget(QtGui.QWidget())
        newtab_layout = QtGui.QVBoxLayout(newtab_2.widget())
        newtab_2.setWidgetResizable(True)
        global redit
        redit = {}
        #Auto completer
        c = QtGui.QCompleter(["fixedValue", "zeroGradient"])
        c.setCompletionMode(QtGui.QCompleter.UnfilteredPopupCompletion)
        d = QtGui.QCompleter(['uniform (x y z)', 'uniform p'])
        d.setCompletionMode(QtGui.QCompleter.UnfilteredPopupCompletion)
        #grid positioner
        grid = QtGui.QGridLayout()
        font = QtGui.QFont()
        font.setUnderline(True)
        font.setBold(True)
        n = 1
        for f in turbprop:
            lable = QtGui.QLabel(str(f))
            lable.setFont(font)
            grid.addWidget(lable,0,n)
            n += 1
        font.setUnderline(False)
        # Counters to keep control on boxes
        j = 1
        i = 1
        while len(first) > 0:
            label = QtGui.QLabel(first.pop())
            label.setFont(font)
            grid.addWidget(label, j, 0)
            label = QtGui.QLabel("Type: ")
            grid.addWidget(label, j+1, 0)
            label = QtGui.QLabel("Value: ")
            grid.addWidget(label, j+2, 0)
            for x in range(1, n):
                redit[(i)] = QtGui.QComboBox()
                grid.addWidget(redit[(i)], j+1, x)
                redit[(i+1)] = QtGui.QComboBox()
                grid.addWidget(redit[(i+1)], j+2, x)
                redit[(i)].setEditable(True)
                redit[(i+1)].setEditable(True)
                redit[(i)].setCompleter(c)
                redit[(i+1)].setCompleter(d)
                i += 2
            j += 3
        newtab_2.setLayout(grid)
        #Create a new tab, then switch
        self.Schemes.addTab(newtab_2, "Turbulence Properties")

```

```python
    #Set the options nedded for turbulence
    def setturbulence(self):
        ras = case+"/constant/RASProperties"
        replace.replace(ras, "laminar;", str(self.comboBox_4.currentText())+";")
        replace.replace(ras, "off;", "on;")
        tur = case+"/constant/turbulenceProperties"
        replace.replace(tur, "nu;", self.textEdit.toPlainText()+";")
        for f in turbprop:
            shutil.copy2(case+"/templates/"+f+"_tmp", case+"/"+str(start)+"/"+f)
        i = 1
        for j in range(con-1, -1, -1):
            for f in turbprop:
                var = case+"/"+str(start)+"/"+str(f)
                typ = redit[(i)].currentText()
                val = redit[(i+1)].currentText()
                replace.replace(var, "typ"+str(j)+"/*type*/;", str(typ)+";")
                if typ == "zeroGradient":
                    replace.replace(var, "value           val"+str(j)+"/*val*/;", "")
                else:
                    replace.replace(var, "val"+str(j)+"/*val*/;", str(val)+";")
                i += 2

    #Remove turbulence related files and tabs
    def noturbulence(self):
        self.textEdit.setEnabled(False)
        self.label_14.setEnabled(False)
        self.comboBox_4.setEnabled(False)
        shutil.copy2(case+"/templates/RASProperties", case+"/constant/RASProperties")
        self.Schemes.removeTab(7)

    #Setting the viscosity
    def visc(self):
        shutil.copy2(case+"/templates/transportProperties", case+"/constant/transportProperties")
        vis = case+"/constant/transportProperties"
        replace.replace(vis, "nu;", self.textEdit_10.toPlainText()+";")

    #Allows for additional terminal commands
    def terminal(self):
        try:
            case
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No case defined", QtGui.QMessageBox.Ok)
            return
        command = str(self.lineEdit_2.text())
        command = command.split(' ')
        self.lineEdit_2.clear()
        cmd = open(pwd+"/outp/terminal","w")
        if len(command) == 1:
```

```
            t = subprocess.Popen([command[0]], cwd=case, stdout = cmd, stderr = cmd)
        elif len(command) == 2:
            t = subprocess.Popen([command[0], command[1]], cwd=case, stdout = cmd, stderr = cmd)
        elif len(command) == 3:
            t = subprocess.Popen([command[0], command[1], command[2]], cwd=case, stdout = cmd, stderr = cmd)
        else:
            self.textEdit_9.append("Command not recognized")
        try:
            t
        except:
            return
        t.wait()
        cmd.close()
        self.textEdit_9.append(open(pwd+"/outp/terminal").read())

    #Launches paraFOAM
    def para(self):
        try:
            case
        except:
            QtGui.QMessageBox.question(self, 'An error has occured', "No case defined", QtGui.QMessageBox.Ok)
            return
        e = subprocess.Popen(["paraFoam"], cwd = case)

    #Checks if input is a float/number
    def validate(self):
        if not self.isFloat(self.textEdit.toPlainText(), "Viscosity is ill defined"):
            return False
        if not self.isFloat(self.textEdit_2.toPlainText(), "Pressure tolerace is ill defined"):
            return False
        if not self.isFloat(self.textEdit_3.toPlainText(), "Velocity  tolerace is ill defined"):
            return False
        if not self.isFloat(self.textEdit_4.toPlainText(), "Lenght of simulation is ill defined"):
            return False
        if not self.isFloat(self.textEdit_5.toPlainText(), "Timestep  is ill defined"):
            return False
        if not self.isFloat(self.textEdit_6.toPlainText(), "Write interval  is ill defined"):
            return False
        if not self.isFloat(self.textEdit_7.toPlainText(), "Pressure relTol is ill defined"):
            return False
        if not self.isFloat(self.textEdit_8.toPlainText(), "Velocity relTol is ill defined"):
            return False
        if not self.isFloat(self.textEdit_12.toPlainText(), "Starttime is ill defined"):
            return False
```

```python

            return True

    #Checks if parameter is a float
    def isFloat(self, inp, text):
        try:
            float(inp)
            return True
        except ValueError:
            QtGui.QMessageBox.question(self, 'An error has occured',
                text, QtGui.QMessageBox.Ok)
            return False

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    window = Main()
    window.show()
    sys.exit(app.exec_())
```

Appendix B: Interface

```
# -*- coding: utf-8 -*-
'''
GUI for openFOAM

This is the main interface, generated by QT Designer

Requirements: openFOAM 2.2.2 with paraFOAM running on UBUNTU
Made by: Henrik Kaald Melb\o
NTNU, Trondheim, Norway
Department Of Chemical Engineering
Contact: henrikkaald@gmail.com
Version: 3.0
Date: 14. Mar. 2014
'''

from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig,
            _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Ui_openFOAM(object):
    def setupUi(self, openFOAM):
        openFOAM.setObjectName(_fromUtf8("openFOAM"))
        openFOAM.resize(844, 641)
        self.gridLayout = QtGui.QGridLayout(openFOAM)
        self.gridLayout.setObjectName(_fromUtf8("gridLayout"))
        self.buttonBox = QtGui.QDialogButtonBox(openFOAM)
```

```
        self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
        self.buttonBox.setStandardButtons(QtGui.QDialogButtonBox.Apply|QtGui.QDialogButtonBox.Cancel)
        self.buttonBox.setObjectName(_fromUtf8("buttonBox"))
        self.gridLayout.addWidget(self.buttonBox, 2, 0, 1, 1, QtCore.Qt.AlignHCenter)
        self.Schemes = QtGui.QTabWidget(openFOAM)
        self.Schemes.setObjectName(_fromUtf8("Schemes"))
        self.Solver = QtGui.QWidget()
        self.Solver.setObjectName(_fromUtf8("Solver"))
        self.comboBox = QtGui.QComboBox(self.Solver)
        self.comboBox.setGeometry(QtCore.QRect(40, 150, 311, 29))
        self.comboBox.setObjectName(_fromUtf8("comboBox"))
        self.comboBox.addItem(_fromUtf8(""))
        self.pushButton_2 = QtGui.QPushButton(self.Solver)
        self.pushButton_2.setGeometry(QtCore.QRect(290, 360, 98, 31))
        self.pushButton_2.setObjectName(_fromUtf8("pushButton_2"))
        self.pushButton_3 = QtGui.QPushButton(self.Solver)
        self.pushButton_3.setGeometry(QtCore.QRect(430, 360, 98, 31))
        self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
        self.pushButton_4 = QtGui.QPushButton(self.Solver)
        self.pushButton_4.setGeometry(QtCore.QRect(360, 440, 98, 31))
        self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
        self.textEdit_11 = QtGui.QTextEdit(self.Solver)
        self.textEdit_11.setGeometry(QtCore.QRect(440, 70, 311, 241))
        self.textEdit_11.setObjectName(_fromUtf8("textEdit_11"))
        self.Schemes.addTab(self.Solver, _fromUtf8(""))
        self.tab_2 = QtGui.QWidget()
        self.tab_2.setObjectName(_fromUtf8("tab_2"))
        self.pushButton = QtGui.QPushButton(self.tab_2)
        self.pushButton.setGeometry(QtCore.QRect(570, 220, 98, 31))
        self.pushButton.setObjectName(_fromUtf8("pushButton"))
        self.lineEdit = QtGui.QLineEdit(self.tab_2)
        self.lineEdit.setGeometry(QtCore.QRect(30, 220, 511, 33))
        self.lineEdit.setObjectName(_fromUtf8("lineEdit"))
        self.label_18 = QtGui.QLabel(self.tab_2)
        self.label_18.setGeometry(QtCore.QRect(40, 160, 391, 21))
        self.label_18.setObjectName(_fromUtf8("label_18"))
        self.checkBox_2 = QtGui.QCheckBox(self.tab_2)
        self.checkBox_2.setGeometry(QtCore.QRect(40, 310, 181, 26))
        self.checkBox_2.setObjectName(_fromUtf8("checkBox_2"))
        self.Schemes.addTab(self.tab_2, _fromUtf8(""))
        self.Controls = QtGui.QWidget()
        self.Controls.setObjectName(_fromUtf8("Controls"))
        self.pushButton_5 = QtGui.QPushButton(self.Controls)
        self.pushButton_5.setGeometry(QtCore.QRect(240, 240, 98, 31))
        self.pushButton_5.setObjectName(_fromUtf8("pushButton_5"))
        self.pushButton_7 = QtGui.QPushButton(self.Controls)
        self.pushButton_7.setGeometry(QtCore.QRect(490, 240, 98, 31))
        self.pushButton_7.setObjectName(_fromUtf8("pushButton_7"))
        self.Schemes.addTab(self.Controls, _fromUtf8(""))
        self.fvSchemes = QtGui.QWidget()
        self.fvSchemes.setObjectName(_fromUtf8("fvSchemes"))
        self.comboBox_5 = QtGui.QComboBox(self.fvSchemes)
        self.comboBox_5.setGeometry(QtCore.QRect(260, 70, 211, 29))
        self.comboBox_5.setObjectName(_fromUtf8("comboBox_5"))
        self.comboBox_5.addItem(_fromUtf8(""))
```

```
        self.comboBox_5.addItem(_fromUtf8(""))
        self.comboBox_5.addItem(_fromUtf8(""))
        self.comboBox_5.addItem(_fromUtf8(""))
        self.comboBox_5.addItem(_fromUtf8(""))
        self.comboBox_6 = QtGui.QComboBox(self.fvSchemes)
        self.comboBox_6.setGeometry(QtCore.QRect(260, 110, 211, 29))
        self.comboBox_6.setObjectName(_fromUtf8("comboBox_6"))
        self.comboBox_6.addItem(_fromUtf8(""))
        self.comboBox_6.addItem(_fromUtf8(""))
        self.comboBox_6.addItem(_fromUtf8(""))
        self.comboBox_6.addItem(_fromUtf8(""))
        self.comboBox_6.addItem(_fromUtf8(""))
        self.comboBox_6.addItem(_fromUtf8(""))
        self.comboBox_7 = QtGui.QComboBox(self.fvSchemes)
        self.comboBox_7.setGeometry(QtCore.QRect(260, 150, 211, 29))
        self.comboBox_7.setObjectName(_fromUtf8("comboBox_7"))
        self.comboBox_7.addItem(_fromUtf8(""))
        self.comboBox_7.addItem(_fromUtf8(""))
        self.comboBox_7.addItem(_fromUtf8(""))
        self.comboBox_7.addItem(_fromUtf8(""))
        self.comboBox_8 = QtGui.QComboBox(self.fvSchemes)
        self.comboBox_8.setGeometry(QtCore.QRect(260, 310, 211, 29))
        self.comboBox_8.setObjectName(_fromUtf8("comboBox_8"))
        self.comboBox_8.addItem(_fromUtf8(""))
        self.comboBox_8.addItem(_fromUtf8(""))
        self.comboBox_8.addItem(_fromUtf8(""))
        self.comboBox_8.addItem(_fromUtf8(""))
        self.comboBox_8.addItem(_fromUtf8(""))
        self.comboBox_10 = QtGui.QComboBox(self.fvSchemes)
        self.comboBox_10.setGeometry(QtCore.QRect(260, 390, 211, 29))
        self.comboBox_10.setObjectName(_fromUtf8("comboBox_10"))
        self.comboBox_10.addItem(_fromUtf8(""))
        self.comboBox_10.addItem(_fromUtf8(""))
        self.comboBox_10.addItem(_fromUtf8(""))
        self.comboBox_10.addItem(_fromUtf8(""))
        self.textEdit_2 = QtGui.QTextEdit(self.fvSchemes)
        self.textEdit_2.setGeometry(QtCore.QRect(260, 190, 91, 31))
        self.textEdit_2.setObjectName(_fromUtf8("textEdit_2"))
        self.textEdit_3 = QtGui.QTextEdit(self.fvSchemes)
        self.textEdit_3.setGeometry(QtCore.QRect(260, 430, 91, 31))
        self.textEdit_3.setObjectName(_fromUtf8("textEdit_3"))
        self.label_4 = QtGui.QLabel(self.fvSchemes)
        self.label_4.setGeometry(QtCore.QRect(40, 30, 66, 21))
        self.label_4.setObjectName(_fromUtf8("label_4"))
        self.label_5 = QtGui.QLabel(self.fvSchemes)
        self.label_5.setGeometry(QtCore.QRect(30, 270, 66, 21))
        self.label_5.setObjectName(_fromUtf8("label_5"))
        self.label_6 = QtGui.QLabel(self.fvSchemes)
        self.label_6.setGeometry(QtCore.QRect(170, 70, 51, 21))
        self.label_6.setObjectName(_fromUtf8("label_6"))
        self.label_7 = QtGui.QLabel(self.fvSchemes)
        self.label_7.setGeometry(QtCore.QRect(180, 310, 51, 21))
        self.label_7.setObjectName(_fromUtf8("label_7"))
        self.label_8 = QtGui.QLabel(self.fvSchemes)
        self.label_8.setGeometry(QtCore.QRect(120, 110, 111, 21))
        self.label_8.setObjectName(_fromUtf8("label_8"))
        self.label_9 = QtGui.QLabel(self.fvSchemes)
```

```
        self.label_9.setGeometry(QtCore.QRect(130, 350, 111, 21))
        self.label_9.setObjectName(_fromUtf8("label_9"))
        self.label_10 = QtGui.QLabel(self.fvSchemes)
        self.label_10.setGeometry(QtCore.QRect(150, 150, 81, 21))
        self.label_10.setObjectName(_fromUtf8("label_10"))
        self.label_11 = QtGui.QLabel(self.fvSchemes)
        self.label_11.setGeometry(QtCore.QRect(160, 390, 81, 21))
        self.label_11.setObjectName(_fromUtf8("label_11"))
        self.label_12 = QtGui.QLabel(self.fvSchemes)
        self.label_12.setGeometry(QtCore.QRect(150, 200, 71, 21))
        self.label_12.setObjectName(_fromUtf8("label_12"))
        self.label_13 = QtGui.QLabel(self.fvSchemes)
        self.label_13.setGeometry(QtCore.QRect(150, 430, 71, 21))
        self.label_13.setObjectName(_fromUtf8("label_13"))
        self.comboBox_9 = QtGui.QComboBox(self.fvSchemes)
        self.comboBox_9.setGeometry(QtCore.QRect(260, 350, 211, 29))
        self.comboBox_9.setObjectName(_fromUtf8("comboBox_9"))
        self.comboBox_9.addItem(_fromUtf8(""))
        self.comboBox_9.addItem(_fromUtf8(""))
        self.comboBox_9.addItem(_fromUtf8(""))
        self.comboBox_9.addItem(_fromUtf8(""))
        self.comboBox_9.addItem(_fromUtf8(""))
        self.comboBox_9.addItem(_fromUtf8(""))
        self.label_23 = QtGui.QLabel(self.fvSchemes)
        self.label_23.setGeometry(QtCore.QRect(170, 240, 51, 21))
        self.label_23.setObjectName(_fromUtf8("label_23"))
        self.label_24 = QtGui.QLabel(self.fvSchemes)
        self.label_24.setGeometry(QtCore.QRect(170, 480, 51, 21))
        self.label_24.setObjectName(_fromUtf8("label_24"))
        self.textEdit_7 = QtGui.QTextEdit(self.fvSchemes)
        self.textEdit_7.setGeometry(QtCore.QRect(260, 230, 91, 31))
        self.textEdit_7.setObjectName(_fromUtf8("textEdit_7"))
        self.textEdit_8 = QtGui.QTextEdit(self.fvSchemes)
        self.textEdit_8.setGeometry(QtCore.QRect(260, 470, 91, 31))
        self.textEdit_8.setObjectName(_fromUtf8("textEdit_8"))
        self.Schemes.addTab(self.fvSchemes, _fromUtf8(""))
        self.fvSolution = QtGui.QWidget()
        self.fvSolution.setObjectName(_fromUtf8("fvSolution"))
        self.label_15 = QtGui.QLabel(self.fvSolution)
        self.label_15.setGeometry(QtCore.QRect(50, 200, 151, 21))
        self.label_15.setObjectName(_fromUtf8("label_15"))
        self.label_16 = QtGui.QLabel(self.fvSolution)
        self.label_16.setGeometry(QtCore.QRect(130, 260, 71, 21))
        self.label_16.setObjectName(_fromUtf8("label_16"))
        self.label_17 = QtGui.QLabel(self.fvSolution)
        self.label_17.setGeometry(QtCore.QRect(100, 320, 111, 21))
        self.label_17.setObjectName(_fromUtf8("label_17"))
        self.textEdit_4 = QtGui.QTextEdit(self.fvSolution)
        self.textEdit_4.setGeometry(QtCore.QRect(230, 190, 104, 31))
        self.textEdit_4.setObjectName(_fromUtf8("textEdit_4"))
        self.textEdit_5 = QtGui.QTextEdit(self.fvSolution)
        self.textEdit_5.setGeometry(QtCore.QRect(230, 250, 104, 31))
        self.textEdit_5.setObjectName(_fromUtf8("textEdit_5"))
        self.textEdit_6 = QtGui.QTextEdit(self.fvSolution)
        self.textEdit_6.setGeometry(QtCore.QRect(230, 310, 104, 31))
        self.textEdit_6.setObjectName(_fromUtf8("textEdit_6"))
        self.label_20 = QtGui.QLabel(self.fvSolution)
```

```
        self.label_20.setGeometry(QtCore.QRect(120, 140, 81, 21))
        self.label_20.setObjectName(_fromUtf8("label_20"))
        self.textEdit_12 = QtGui.QTextEdit(self.fvSolution)
        self.textEdit_12.setGeometry(QtCore.QRect(230, 130, 104, 31))
        self.textEdit_12.setObjectName(_fromUtf8("textEdit_12"))
        self.Schemes.addTab(self.fvSolution, _fromUtf8(""))
        self.tab = QtGui.QWidget()
        self.tab.setObjectName(_fromUtf8("tab"))
        self.checkBox = QtGui.QCheckBox(self.tab)
        self.checkBox.setGeometry(QtCore.QRect(30, 20, 191, 26))
        self.checkBox.setObjectName(_fromUtf8("checkBox"))
        self.textEdit = QtGui.QTextEdit(self.tab)
        self.textEdit.setGeometry(QtCore.QRect(230, 120, 151, 31))
        self.textEdit.setObjectName(_fromUtf8("textEdit"))
        self.comboBox_4 = QtGui.QComboBox(self.tab)
        self.comboBox_4.setGeometry(QtCore.QRect(200, 70, 201, 29))
        self.comboBox_4.setObjectName(_fromUtf8("comboBox_4"))
        self.comboBox_4.addItem(_fromUtf8(""))
        self.comboBox_4.addItem(_fromUtf8(""))
        self.label_14 = QtGui.QLabel(self.tab)
        self.label_14.setGeometry(QtCore.QRect(65, 120, 141, 21))
        self.label_14.setObjectName(_fromUtf8("label_14"))
        self.scrollArea = QtGui.QScrollArea(self.tab)
        self.scrollArea.setGeometry(QtCore.QRect(10, 240, 731, 201))
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setObjectName(_fromUtf8("scrollArea"))
        self.scrollAreaWidgetContents = QtGui.QWidget()
        self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0,
            729, 199))
        self.scrollAreaWidgetContents.setObjectName(_fromUtf8("
            scrollAreaWidgetContents"))
        self.textEdit_9 = QtGui.QTextEdit(self.
            scrollAreaWidgetContents)
        self.textEdit_9.setGeometry(QtCore.QRect(0, 0, 731, 201))
        self.textEdit_9.setObjectName(_fromUtf8("textEdit_9"))
        self.scrollArea.setWidget(self.scrollAreaWidgetContents)
        self.lineEdit_2 = QtGui.QLineEdit(self.tab)
        self.lineEdit_2.setGeometry(QtCore.QRect(10, 460, 551, 33))
        self.lineEdit_2.setObjectName(_fromUtf8("lineEdit_2"))
        self.pushButton_6 = QtGui.QPushButton(self.tab)
        self.pushButton_6.setGeometry(QtCore.QRect(610, 460, 98, 31))
        self.pushButton_6.setObjectName(_fromUtf8("pushButton_6"))
        self.label_19 = QtGui.QLabel(self.tab)
        self.label_19.setGeometry(QtCore.QRect(140, 180, 66, 21))
        self.label_19.setObjectName(_fromUtf8("label_19"))
        self.textEdit_10 = QtGui.QTextEdit(self.tab)
        self.textEdit_10.setGeometry(QtCore.QRect(230, 180, 151, 31))
        self.textEdit_10.setObjectName(_fromUtf8("textEdit_10"))
        self.Schemes.addTab(self.tab, _fromUtf8(""))
        self.Boundary_Conditions = QtGui.QWidget()
        self.Boundary_Conditions.setObjectName(_fromUtf8("
            Boundary_Conditions"))
        self.Schemes.addTab(self.Boundary_Conditions, _fromUtf8(""))
        self.gridLayout.addWidget(self.Schemes, 1, 0, 1, 1)

        self.retranslateUi(openFOAM)
        self.Schemes.setCurrentIndex(0)
```

```
        QtCore.QObject.connect(self.buttonBox, QtCore.SIGNAL(_fromUtf8("accepted()")), openFOAM.accept)
        QtCore.QObject.connect(self.buttonBox, QtCore.SIGNAL(_fromUtf8("rejected()")), openFOAM.reject)
        QtCore.QMetaObject.connectSlotsByName(openFOAM)

    def retranslateUi(self, openFOAM):
        openFOAM.setWindowTitle(_translate("openFOAM", "openFOAM", None))
        self.comboBox.setItemText(0, _translate("openFOAM", "None", None))
        self.pushButton_2.setText(_translate("openFOAM", "New Case", None))
        self.pushButton_3.setText(_translate("openFOAM", "Load Case", None))
        self.pushButton_4.setText(_translate("openFOAM", "paraFOAM", None))
        self.textEdit_11.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-size:12pt;\">Use the combobox to choose an appropriate solver</span></p></body></html>", None))
        self.Schemes.setTabText(self.Schemes.indexOf(self.Solver), _translate("openFOAM", "Solver", None))
        self.pushButton.setText(_translate("openFOAM", "Load", None))
        self.lineEdit.setText(_translate("openFOAM", "Path", None))
        self.label_18.setText(_translate("openFOAM", "Import Mesh", None))
        self.checkBox_2.setText(_translate("openFOAM", "Suply  mesh manually", None))
        self.Schemes.setTabText(self.Schemes.indexOf(self.tab_2), _translate("openFOAM", "Mesh", None))
        self.pushButton_5.setText(_translate("openFOAM", "fvSchemes", None))
        self.pushButton_7.setText(_translate("openFOAM", "fvSolution", None))
        self.Schemes.setTabText(self.Schemes.indexOf(self.Controls), _translate("openFOAM", "     fv     ", None))
        self.comboBox_5.setItemText(0, _translate("openFOAM", "GAMG", None))
        self.comboBox_5.setItemText(1, _translate("openFOAM", "PBiCG", None))
        self.comboBox_5.setItemText(2, _translate("openFOAM", "DIC", None))
        self.comboBox_5.setItemText(3, _translate("openFOAM", "diagonal", None))
        self.comboBox_5.setItemText(4, _translate("openFOAM", "smoothSolver", None))
        self.comboBox_6.setItemText(0, _translate("openFOAM", "DILU", None))
```

```
        self.comboBox_6.setItemText(1, _translate("openFOAM", "None", None))
        self.comboBox_6.setItemText(2, _translate("openFOAM", "DIC", None))
        self.comboBox_6.setItemText(3, _translate("openFOAM", "FDIC", None))
        self.comboBox_6.setItemText(4, _translate("openFOAM", "GAMG", None))
        self.comboBox_6.setItemText(5, _translate("openFOAM", "diagonal", None))
        self.comboBox_7.setItemText(0, _translate("openFOAM", "GaussSeidel", None))
        self.comboBox_7.setItemText(1, _translate("openFOAM", "None", None))
        self.comboBox_7.setItemText(2, _translate("openFOAM", "DIC", None))
        self.comboBox_7.setItemText(3, _translate("openFOAM", "DICGaussSeidel", None))
        self.comboBox_8.setItemText(0, _translate("openFOAM", "GAMG", None))
        self.comboBox_8.setItemText(1, _translate("openFOAM", "PBiCG", None))
        self.comboBox_8.setItemText(2, _translate("openFOAM", "DIC", None))
        self.comboBox_8.setItemText(3, _translate("openFOAM", "diagonal", None))
        self.comboBox_8.setItemText(4, _translate("openFOAM", "smoothSolver", None))
        self.comboBox_10.setItemText(0, _translate("openFOAM", "GaussSeidel", None))
        self.comboBox_10.setItemText(1, _translate("openFOAM", "None", None))
        self.comboBox_10.setItemText(2, _translate("openFOAM", "DIC", None))
        self.comboBox_10.setItemText(3, _translate("openFOAM", "DICGaussSeidel", None))
        self.textEdit_2.setToolTip(_translate("openFOAM", "<html><head/><body><p><span style=\" font-family:\'arial,sans-serif\'; font-size:12px; color:#000000; background-color:#ffffff;\">Solver tolerance</span></p></body></html>", None))
        self.textEdit_2.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">0.00001</p></body></html>", None))
        self.textEdit_3.setToolTip(_translate("openFOAM", "<html><head/><body><p><span style=\" font-family:\'arial,sans-serif\'; font-size:12px; color:#000000; background-color:#ffffff;\">Solver tolerance</span></p></body></html>", None))
```

```
        self.textEdit_3.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">0.00001</p></body></html>", None))
        self.label_4.setText(_translate("openFOAM", "Pressure", None))
        self.label_5.setText(_translate("openFOAM", "Velocity\n"
"", None))
        self.label_6.setText(_translate("openFOAM", "Solver:", None))
        self.label_7.setText(_translate("openFOAM", "Solver:", None))
        self.label_8.setText(_translate("openFOAM", "Preconditioner:", None))
        self.label_9.setText(_translate("openFOAM", "Preconditioner:", None))
        self.label_10.setText(_translate("openFOAM", "Smoother:", None))
        self.label_11.setText(_translate("openFOAM", "Smoother:", None))
        self.label_12.setText(_translate("openFOAM", "Tolerance:", None))
        self.label_13.setText(_translate("openFOAM", "Tolerance:", None))
        self.comboBox_9.setItemText(0, _translate("openFOAM", "DILU", None))
        self.comboBox_9.setItemText(1, _translate("openFOAM", "None", None))
        self.comboBox_9.setItemText(2, _translate("openFOAM", "DIC", None))
        self.comboBox_9.setItemText(3, _translate("openFOAM", "FDIC", None))
        self.comboBox_9.setItemText(4, _translate("openFOAM", "GAMG", None))
        self.comboBox_9.setItemText(5, _translate("openFOAM", "diagonal", None))
        self.label_23.setText(_translate("openFOAM", "Reltol:", None))
        self.label_24.setText(_translate("openFOAM", "Reltol:", None))
        self.textEdit_7.setToolTip(_translate("openFOAM", "<html><head/><body><p><span style=\" font-family:\'arial,sans-serif\'; font-size:12px; color:#000000; background-color:#ffffff;\">Solver relative  tolerance</span></p></body></html>", None))
        self.textEdit_7.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">0.01</p
```

```
></body></html>", None))
        self.textEdit_8.setToolTip(_translate("openFOAM", "<html><head/><body><p><span style=\" font-family:\'arial,sans-serif\'; font-size:12px; color:#000000; background-color:#ffffff;\">Solver relative  tolerance</span></p></body></html>", None))
        self.textEdit_8.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">0.01</p></body></html>", None))
        self.Schemes.setTabText(self.Schemes.indexOf(self.fvSchemes), _translate("openFOAM", "fvSolutions", None))
        self.label_15.setText(_translate("openFOAM", "Lenght of simulation:", None))
        self.label_16.setText(_translate("openFOAM", "Timestep:", None))
        self.label_17.setText(_translate("openFOAM", "Write interval:", None))
        self.textEdit_4.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">1</p></body></html>", None))
        self.textEdit_5.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">0.01</p></body></html>", None))
        self.textEdit_6.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">10</p></
```

```
body></html>", None))
        self.label_20.setText(_translate("openFOAM", "Start time:", None))
        self.textEdit_12.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">0</p></body></html>", None))
        self.Schemes.setTabText(self.Schemes.indexOf(self.fvSolution), _translate("openFOAM", "ControlDict", None))
        self.checkBox.setText(_translate("openFOAM", "Turbulence", None))
        self.textEdit.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">1</p></body></html>", None))
        self.comboBox_4.setItemText(0, _translate("openFOAM", "None", None))
        self.comboBox_4.setItemText(1, _translate("openFOAM", "kEpsilon", None))
        self.label_14.setText(_translate("openFOAM", "Turbulence viscosity:", None))
        self.textEdit_9.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">—Run Additional Terminal Commands—</p>\n"
"<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><br /></p>\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">Helpful suggestions:</p>\n"
"<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><br /></p>\n"
```

```
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">- checkMesh</p>\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">- renumberMesh</p>\n"
"<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><br /></p>\n"
"<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><br /></p></body></html>", None))
        self.pushButton_6.setText(_translate("openFOAM", "Send", None))
        self.label_19.setText(_translate("openFOAM", "Viscosity:", None))
        self.textEdit_10.setHtml(_translate("openFOAM", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'Cantarell\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">1</p></body></html>", None))
        self.Schemes.setTabText(self.Schemes.indexOf(self.tab), _translate("openFOAM", "Properties", None))
        self.Schemes.setTabText(self.Schemes.indexOf(self.Boundary_Conditions), _translate("openFOAM", "Boundary Conditions", None))


if __name__ == "__main__":
    import sys
    app = QtGui.QApplication(sys.argv)
    openFOAM = QtGui.QDialog()
    ui = Ui_openFOAM()
    ui.setupUi(openFOAM)
    openFOAM.show()
    sys.exit(app.exec_())
```

Appendix C: Utility Functions

```
##################################################
#TextEditor for manual edit of fvSchemes         #
##################################################
from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt
from PyQt4.QtGui import *
class TextEditer(QtGui.QDialog):

    def __init__(self, case):
        global place
        place = case
        super(TextEditer, self).__init__()
        self.Ui()

    def Ui(self):

        self.text_edit = QTextEdit(self)
        self.setGeometry(0,0,900,600)
        self.setFixedSize(900, 600)
        self.setWindowTitle('fvSchemes')
        self.text_edit.setGeometry(QtCore.QRect(0, 20, 900, 540))
        text = open(place+"/system/fvSchemes").read()
        self.text_edit.setText(text)
        self.close = QtGui.QPushButton('Close', self)
        self.close.move(350, 565)
        self.close.clicked.connect(self.Close)

    def Close(self):
        self.saveFile()
        self.deleteLater()

    def saveFile(self):
       f = open(place+"/system/fvSchemes", 'w')
       filedata = self.text_edit.toPlainText()
       f.write(filedata)
       f.close()
```

```
##################################################
```

```
#A simple parser for editing text in files            #
#####################################################
import sys, os, time, subprocess, re, shutil
from tempfile import mkstemp
from shutil import move
from os import remove, close
def replace(file_path, pattern, subst):
      #Create temp file
      fh, abs_path = mkstemp()
      new_file = open(abs_path,'w')
      old_file = open(file_path)
      for line in old_file:
          new_file.write(line.replace(pattern, subst))
      #close temp file
      new_file.close()
      close(fh)
      old_file.close()
      #Remove original file
      remove(file_path)
      #Move new file
      move(abs_path, file_path)
```

```
#####################################################
#Loadbar: While program is running                  #
#####################################################
from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt
from PyQt4.QtGui import *
import sys, os, time, subprocess, re, shutil
from os import remove, close
class Loader(QtGui.QDialog):
    def __init__(self, case, run, delta, write, program, pwd, start):
        global place, ru, delt, wri, sta
        ru = run
        sta = start
        delt = delta
        wri = write
        place = case
        self.prog(program, pwd, case)
        super(Loader, self).__init__()
        self.pbar = QtGui.QProgressBar(self)
        self.pbar.setGeometry(30, 40, 200, 25)
        self.setGeometry(800, 430, 280, 150)
        self.btn = QtGui.QPushButton('Cancel', self)
        self.btn.move(80, 80)
        self.btn.clicked.connect(self.doAction)
        self.step = 0
        self.setWindowTitle('Running')
        #setup timer
        self.timer = QtCore.QTimer(self)
        self.timer.timeout.connect(self.Time)
        self.completed = False
        self.term = False

    def prog(self, program, pwd, case):
        fh = open(pwd+"/outp/Output","w")
        global p
```

```
        p = subprocess.Popen([program], cwd=case, stdout = fh, stderr = fh)
        fh.close()

    def Run(self):
        self.timer.start(100)
        return self.exec_()

    def Complete(self, completed):
        if self.completed:
            return
        self.completed = True
        self.timer.stop()
        if p.poll() == None:
            p.kill()
            completed = False
        if completed:
            self.accept()
        else:
            self.reject()

    def Time(self):
        run2 = re.findall("\d+.\d+|\d+", ru)
        delta2 = re.findall("\d+.\d+|\d+", delt)
        write2 = re.findall("\d+.\d+|\d+", wri)
        start2 = re.findall("\d+.\d+|\d+", sta)
        run3 = float(run2.pop())
        delta3 = float(delta2.pop())
        write3 = float(write2.pop())
        start3 = float(start2.pop())
        tot = (run3-start3)/delta3/write3
        #Counts the number of folders in case folder
        x = subprocess.Popen("ls -d */", bufsize = 1, stdin = subprocess.PIPE, stdout = subprocess.PIPE, stderr = subprocess.PIPE, cwd = place, shell = True)
        count = 0
        #Gives a % of completed "folders"
        if x.stdout:
            for line in x.stdout:
                count += 1
        if count == 4:
            self.step = 0
        else:
            self.step = (float(count-4)/tot)*100
        self.pbar.setValue(self.step)
        QtGui.qApp.processEvents()
        if p.poll() is not None:
            self.Complete(True)
        if self.term == True:
            p.kill()
            self.Complete(False)

    def doAction(self):
        #Kills the process if cancel is pressed
        self.term = True
        p.kill()
        self.deleteLater()
```

```python
#########################################################
#TextEditor for manual edit of fvSchemes               #
#########################################################
from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt
from PyQt4.QtGui import *
class fvSo(QtGui.QDialog):

    def __init__(self, case):
        global place
        place = case
        super(fvSo, self).__init__()
        self.Ui()

    def Ui(self):

        self.text_edit = QTextEdit(self)
        self.setGeometry(0,0,900,600)
        self.setFixedSize(900, 600)
        self.setWindowTitle('fvSolution')
        self.text_edit.setGeometry(QtCore.QRect(0, 20, 900, 540))
        text = open(place+"/system/fvSolution").read()
        self.text_edit.setText(text)
        self.close = QtGui.QPushButton('Close', self)
        self.close.move(350, 565)
        self.close.clicked.connect(self.Close)

    def Close(self):
        self.saveFile()
        self.deleteLater()

    def saveFile(self):
       f = open(place+"/system/fvSolution", 'w')
       filedata = self.text_edit.toPlainText()
       f.write(filedata)
       f.close()
```

```python
#########################################################
#Show Output: After program is done                    #
#########################################################
from PyQt4 import QtGui, QtCore
from PyQt4.QtCore import Qt
from PyQt4.QtGui import *
import sys, os, time, subprocess, re, shutil
class Wiew(QtGui.QDialog):
    def __init__(self, case, pwd):
        global place
        place = case
        pwd = pwd+"/outp"
        super(Wiew, self).__init__()
        self.setGeometry(0,0,900,600)
        self.setFixedSize(900, 600)
        self.text_edit = QTextEdit(self)
        self.text_edit.setGeometry(QtCore.QRect(0, 20, 900, 540))
        text = open(pwd+'/Output').read()
        self.text_edit.setText(text)
```

```
        self.setWindowTitle('Output')
        self.close = QtGui.QPushButton('Close', self)
        self.close.move(250, 565)
        self.close.clicked.connect(self.Close)
        self.save = QtGui.QPushButton('Save', self)
        self.save.move(350, 565)
        self.save.clicked.connect(self.Save)
        self.btn2 = QtGui.QPushButton('paraFOAM', self)
        self.btn2.move(450, 565)
        self.btn2.clicked.connect(self.para)

    def Close(self):
        self.deleteLater()

    def Save(self):
        filename = QtGui.QFileDialog.getSaveFileName(self, 'Save File', os.getenv('HOME'))
        f = open(filename, 'w')
        filedata = self.text_edit.toPlainText()
        f.write(filedata)
        f.close()

    def para(self):
        #opens paraFoam on completion
        e = subprocess.Popen(["paraFoam"], cwd = place)
```

```
####################################################
# A simple parser for creating a set of filenames,#
# used in "findbounds"                             #
####################################################
def parse(path, text, place):
    f = open(path, "r")
    contents = f.readlines()
    f.close

    contents.insert(place, text)

    f = open(path ,"w")
    contents = "".join(contents)
    f.write(contents)
    f.close()
```

Appendix D: Solvers

```
#######################################################
# Defines the class "Solver" used when                #
# implementing other solvers                          #
#######################################################
class Solver:
        name = ""
        description = ""
        properties = ""

        def __init__(self, name, description, properties):
            self.name = name
            self.description = description
            self.properties = properties

def make(name, description, properties):
        solver = Solver(name, description, properties)
        return solver
```

```
#######################################################
# Example of implementing simpleFoam solver                   #
#######################################################
import solver

def simpleFoam():
    s = solver.Solver("simpleFoam",
                      "SimpleFoam is a steady-state solver for incompressible, turbulent flow",
                      {"U", "p"})
    return s
```

```
#######################################################
# Example of implementing icoFoam solver              #
#######################################################
import solver

def icoFoam():
    s = solver.Solver("icoFoam",
```

```
                                "IcoFoam is a transient solver for incompressible, laminar flow of Newtonian fluids",
                                {"U", "p"})
    return s
```

Appendix E: Turbulence

```
########################################################
# Defines the class "turb" used when implementing #
# turbulence models                                #
########################################################
class turb:
        name = ""
        properties = ""

        def __init__(self, name, properties):
            self.name = name
            self.properties = properties

def make(name, properties):
        solver = Solver(name, properties)
        return solver
```

```
########################################################
# Example of implementing kEpsilon turbulence      #
#  model                                           #
########################################################
import turb

def kEpsilon():
    s = turb.turb("kEpsilon",
                    {"k", "epsilon", "nut", "nuTilda"})
    return s
```