Runtime Selection

Models, Boundary Conditions and functionObjects

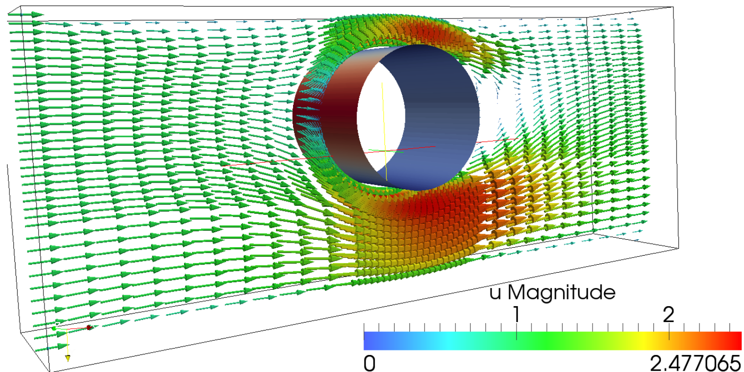
Prof Gavin Tabor

Friday 25th May 2018

CCP-WSI a Collaborative Computational Project in Wave Structure Interaction

University of Exeter

Results : `offsetCylinder` case from tutorials



Run time selectivity

`nonNewtonianIcoFoam` allows selection of which viscosity model we want. How does this work, and can we hang our new model into this framework?

Need to understand how classes work. OOP is about more than designing new language types – it allows us to define relationships between classes.

Two possible ways to program the `complex` class. A complex number can be represented by real and imaginary variables :

```
class complex
{
        //- Real and imaginary parts of the complex number
        scalar re, im;
        .....
```

This is *encapsulation*, a “has-a” relationship

Inheritance

Alternatively, recognise that a complex number is a point on a 2-d plane, with particular extra properties (phase angle, functions such as log. . .).

If we had an existing class `point` we could *extend* this to add extra features :

```
class complex : public point
{
      extra parts go here!!
```

This is *inheritance*, a “is-a” relationship. The new class *extends* the definition of the old one.

Interface vs. Implementation

In practice, the users don't need to know *how* the complex number is represented (the *implementation*) – just what functions they can use. This is the *interface* – defined by the class definition.

We can take this further and define a *virtual base class*, which is just the interface with *no* implementation. Any class derived from this has to define how the various functions work, but will thus have the same interface, and so be interchangeable.

All non-Newtonian viscosity models have to return a value for ν. If we derive them all from a virtual base class, this will force them to have the same interface, so they can be accessed from a list – *run time selection*.

Polymorphism

This is an example of a concept known as *polymorphism* :
"One of the key features of class inheritance is that a pointer to a derived class is type-compatible with a pointer to its base class. *Polymorphism* is the art of taking advantage of this simple but powerful and versatile feature."

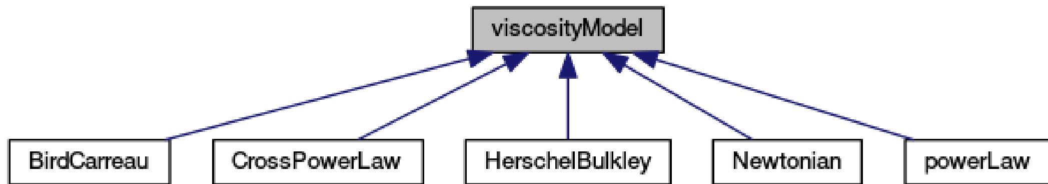
All viscosity models in OpenFOAM are derived from a base class `viscosityModel`. This defines a run time selector function and virtual functions `nu()` and `correct()`

Classes stored in

`/opt/openfoam5/src/transportModels/incompressible/viscosityModels`

and sub-directories

Examples – *viscosityModels*



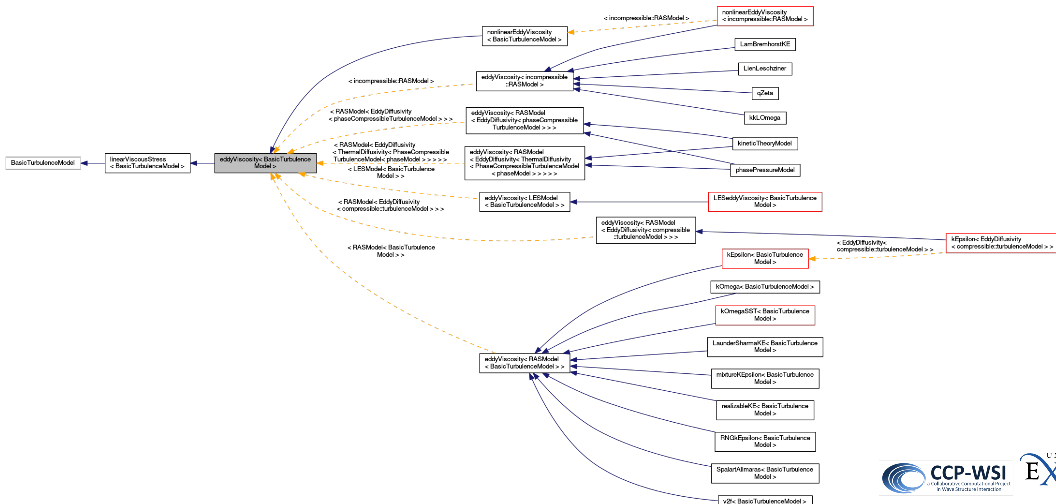
Polymorphism in OpenFOAM

Quite a few things are polymorphic in OpenFOAM;

- Turbulence models
- Boundary conditions
- `functionObjects`

etc. If we want to create a new turbulence model (viscous model, B.C etc), just derive it from the base class and it can plug in alongside any other model. OF even has run time 'hooks' in *controlDict* which mean the code can be added at runtime.

Inheritance – turbulence models



CCP-WSI a Collaborative Computational Project in Wave Structure Interaction

UNIVERSITY OF EXETER

Implementing the Casson model

Easiest (again) to copy existing model – eg. `powerLaw`

1. Copy `powerLaw` sub-directory to home directory
2. Rename the files `Chocolate.H` and `Chocolate.C`; and also change all instances of `powerLaw` to `Chocolate` inside the files.
3. Change over private data to hold the Casson model coefficients; re-implement constructor, `nu()` and `correct()` functions

The make system will compile libraries as well – command `wmake libso`. Again; this uses information from a directory `Make`. Modify the one from `viscosityModels` :

4. `files` needs to read in `Chocolate.C` and write to a library `libChocolateFlowModels` in the user directory
5. `options` needs to reference the `transportModels` library.

CCP-WSI a Collaborative Computational Project in Wave Structure Interaction
UNIVERSITY OF EXETER

files, options

```
<grtabor@sibelius>more files
Chocolate.C
LIB = $(FOAM_USER_LIBBIN)/libChocolateFlowModels
```

```
<grtabor@sibelius>more options
EXE_INC = \
    -I.. \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude

LIB_LIBS = \
    -lfiniteVolume \
    -lincompressibleTransportModels
```

Alterations to case files

Finally need to alter the offsetCylinder case files :

6. Include the line

```
libs ( "libChocolateFlowModels.so" );
```

   in controlDict

7. Specify the coefficients in transportProperties :

```
transportModel Chocolate;
ChocolateCoeffs
{
    eta         eta [ 1 -1 -1 0 0 0 0 ] 4.86;
    tau_y       tau_y [ 1 -1 -2 0 0 0 0 ] 14.38;
    rho         rho [1 -3 0 0 0 0 0] 1200;
}
```

Run offsetCylinder case with nonNewtonianIcoFoam!!

Classes in C++

A *class* in C++ is a structure containing data and functions that act on that data.

These can be

Private – can only be used/manipulated inside the class object

Public – accessible outside

(Protected) ...

Normally, data is `private` and functions (methods) are `public`

When we use a class we *instanciate* an instance of the class – like declaring a variable.

CCP-WSI a Collaborative Computational Project in Wave Structure Interaction

UNIVERSITY OF EXETER

Class functions

Most class functions are called thus :

```
var.myFunc();
```

or

```
ptrVar->myFunc();
```

As a class function, `myFunc()` can access `private` data in `var` – it can also have additional variables passed to it.

In C++, some functions are declared `friend`s – not part of the class but able to access `private` data. Operators (+,- etc) are friend functions.

Class declaration is in `.H` file – actual code in `.C` file (except `inline` functions and `virtual` functions)

Types of class function

Constructor – function called to set up instance of class. This will need to call constructors for any base class(es), and should provide values for any internal variables (no null constructors allowed)

Copy constructor – invoked when a class instance is duplicated. (Sometimes explicitly removed to stop this happening!)

Destructor – usually designated `myClass()` – called when a class instance is deleted; tidies things up

Virtual – function declaration only; implementation in derived class

Access – function to return private data in some form

Run-time database

(One of) the main functions of classes is to partition off data – avoid clashes between different variable names. Visibility of data (and privacy) really important.

However

. . . sometimes we want to break this and access objects out of their scope.

OpenFOAM does this using the object database – function call `.db()`. (Almost) every class includes the object database at some level and this can be interrogated to return any object instance (providing you know its name).

Parabolic Inlet

Laminar flow in a pipe gives a parabolic profile – lets implement a new b.c. for this :

$$\underline{u} = \underline{\hat{n}} \cdot u_m \left(1 - \frac{y^2}{R^2} \right)$$

Process :

1. Identify an existing B.C. to modify
2. Copy across to user working directory
3. Rename files/classes
4. Re-write class functions
5. Set up library compilation and compile
6. Link in runtime and test

Boundary Conditions

B.C. are in *src/finiteVolume/fields/fvPatchFields*; subdirectories *basic*, *constraint*, *derived*, *fvPatchField*

- *fvPatchField* is the (virtual) base class
- *basic* contains intermediate classes; in particular *fixedValue*, *fixedGradient*, *zeroGradient*, *mixed*
- *derived* contains the actual useable classes. *cylindricalInletVelocity* (derived from *fixedValue*) looks suitable!

Initial steps

So :

1. Copy the directory across to the user directory
2. Rename files *cylindricalInletVelocity* → *parabolicInletVelocity* (.C, .H files)
3. Change *cylindrical*→*parabolic* throughout
4. Set up *Make* directory with *files* and *options*
5. Check that it compiles – *wmake libso*

files :

```
parabolicInletVelocityFvPatchVectorField.C

LIB = $(FOAM_USER_LIBBIN)/libnewBC
```

options :

```
EXE_INC = \
    -I$(LIB_SRC)/triSurface/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude

LIB_LIBS = \
    -lOpenFOAM \
    -ltriSurface \
    -lmeshTools \
    -lfiniteVolume
```

CCP-WSI a Collaborative Computational Project in Wave Structure Interaction

UNIVERSITY OF EXETER

Changing the Code

C++ classes contain data, class functions. For *cylindricalInletVelocity* class functions are : various *constructors* (complicated), *updateCoeffs()*, *write(Ostream&)*.

```
class parabolicInletVelocityFvPatchVectorField
:
    public fixedValueFvPatchVectorField
{
    // Private data

        //- Axial velocity
        const scalar maxVelocity_;

        //- Central point
        const vector centre_;

        //- Axis
        const vector axis_;

        //- Radius
        const scalar R_;

public:

   //- Runtime type information
   TypeName("parabolicInletVelocity");
```

Private data : we need vectors for the centre of the inlet and an axis direction (already there) and scalars for the maximum velocity and the pipe radius.

Also need *TypeName* – will become the name of the B.C at run time

Constructor functions

This gets set to zero for a null constructor :

```
Foam::
parabolicInletVelocityFvPatchVectorField::
parabolicInletVelocityFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedValueFvPatchField<vector>(p, iF),
    maxVelocity_(0),
    centre_(pTraits<vector>::zero),
    axis_(pTraits<vector>::zero),
    R_(0)
{}
```

```
Foam::
parabolicInletVelocityFvPatchVectorField::
parabolicInletVelocityFvPatchVectorField
(
    const parabolicInletVelocityFvPatchVectorField& ptf,
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    fixedValueFvPatchField<vector>(ptf, p, iF, mapper),
    maxVelocity_(ptf.maxVelocity_),
    centre_(ptf.centre_),
    axis_(ptf.axis_),
    R_(ptf.R_)
{}
```

. . . and copied across for a copy construct

Read in . . .

We want to read in the actual values from the velocity file – a *dictionary* :

```
Foam::
parabolicInletVelocityFvPatchVectorField::
parabolicInletVelocityFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedValueFvPatchField<vector>(p, iF, dict),
    maxVelocity_(readScalar(dict.lookup("maxVelocity"))),
    centre_(dict.lookup("centre")),
    axis_(dict.lookup("axis")),
    R_(readScalar(dict.lookup("radius")))
{}
```

. . . and write out

Write out through the *write(Ostream& os)* function :

```
void Foam::parabolicInletVelocityFvPatchVectorField::write(Ostream& os) const
{
    fvPatchField<vector>::write(os);
    os.writeKeyword("maxVelocity") << maxVelocity_ <<
        token::END_STATEMENT << nl;
    os.writeKeyword("centre") << centre_ << token::END_STATEMENT << nl;
    os.writeKeyword("axis") << axis_ << token::END_STATEMENT << nl;
    os.writeKeyword("radius") << R_ <<
        token::END_STATEMENT << nl;
    writeEntry("value", os);
}
```

updateCoeffs()

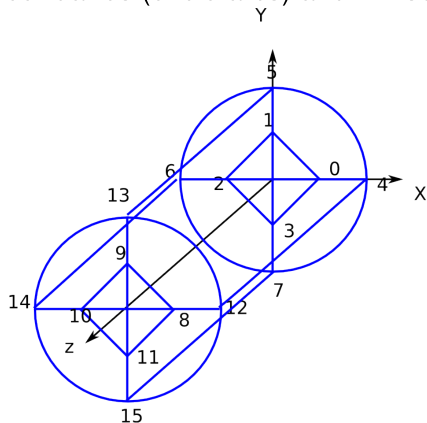
This is the actual code setting the boundary conditions

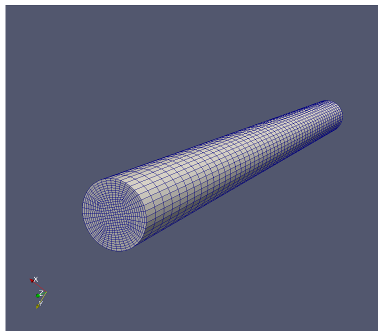
- Again; we can modify what is already there!
- OpenFOAM syntax makes this easier
- (Note call to updateCoeffs() in parent class)

```
void Foam::parabolicInletVelocityFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    vector hatAxis = axis_/mag(axis_);

    const scalarField r(mag(patch().Cf() - centre_));

    operator==(hatAxis*maxVelocity_*(1.0 - (r*r)/(R_*R_)));

    fixedValueFvPatchField<vector>::updateCoeffs();
}
```

Pipe flow case

Set up a test case – flow in a circular pipe. 5 block *blockMesh* demonstrating curved boundaries (circle arcs) and m4 script variables





CCP-WSI
a Collaborative Computational Project in Wave Structure Interaction
UNIVERSITY OF EXETER

B.C syntax

If we look at the "Read in . . ." constructor, see that we need to specify

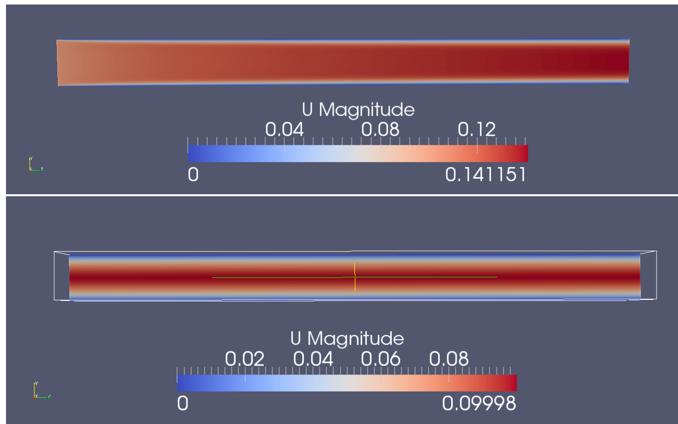
- maxVelocity
- *centre* (a vector)
- *axis* (another vector)
- *radius*

Also need a dummy "*value*"

```
inlet
{
    type            parabolicInletVelocity;
    axis            (0 0 1);
    centre          (0 0 0);
    maxVelocity     30;
    radius          10;
    value           (0 0 0);
}
```

Results

Replace inlet with new condition



Programming Tools

Tools and information sources which may make your life easier!

You *can* do all your programming using command line + editor (such as emacs). However many programmers use IDE – Integrated Development Environment – provides integrated tools (such as highlighting) to make your life easier.

One example – Eclipse. Instructions (thanks Ben J) online :

- `https://openfoamwiki.net/index.php/HowTo_Use_OpenFOAM_with_Eclipse`
- `https://openfoamwiki.net/index.php/HowTo_Use_OpenFOAM_with_Eclipse/Fool_the_indexer`
- `https://www.youtube.com/watch?v=yT9Ia8ESVoY`

CCP-WSI a Collaborative Computational Project in Wave Structure Interaction

UNIVERSITY OF EXETER

doxygen

`doxygen` is a software tool for analysing C++ class structures. It relies on structured comments in header files plus C++ keywords to generate html documentation.

OpenFOAM class files written to take advantage of this – can run `doxygen` on library to generate output.

Results also online;`https://cpp.openfoam.org/v5/`

Useful for identifying class functions, class relationships etc.

Class files

Easiest way to build a new class – start with pre-existing one. Get familiar with what is in the library!

(Easiest way to build new app – start from pre-existing one!)

There are tools for setting up new class files from scratch in the library – `foamNewSource`, (`foamNewTemplate`). Run these to generate a "bare-bones" framework and fill in the spaces.

Other sources

Prof Hakan Nilsson runs an MSc course on OpenFOAM – student project reports online
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/

Dr Joszef Nagy maintains the community repository
https://wiki.openfoam.com/Tutorials

UK&RI Users Group, international OpenFOAM Workshop both feature training days
http://openfoamworkshop.org/

Prof Hrv Jasak runs “The Summer School” (various times of year) – boot camp for OpenFOAM developers! https://foam-extend.fsb.hr/numap/

CCP-WSI a Collaborative Computational Project in Wave Structure Interaction

UNIVERSITY OF EXETER

Summary

Take-home message(s) :

- OpenFOAM programming should not be seen as scary or risky!!
- Think “MatLab for CFD”.
- Easy to read code; modify existing apps; implement transport equations – even add whole new models at run time.

More info : email: g.r.tabor@ex.ac.uk

CCP-WSI a Collaborative Computational Project in Wave Structure Interaction

UNIVERSITY OF EXETER