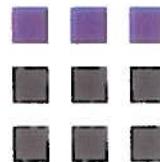


CFD Direct Training Manual

Programming CFD



CFD Direct

Programming CFD

OpenFOAM Training

C.J. Greenshields

Notes v4.1 rev 1. 22/11/2016

Copyright © 2015-2016 CFD Direct Ltd

All rights reserved. Any unauthorised reproduction of any form will constitute an infringement of the copyright.

The right of C.J. Greenshields to be identified as the author of this work has been asserted

CFD Direct Ltd makes no warranty, express or implied, to the accuracy or completeness of the information in this guide and therefore the information in this guide should not be relied upon. CFD Direct Ltd disclaims liability for any loss, howsoever caused, arising directly or indirectly from reliance on the information in this guide.

■ Introduction to programming

1.1 Coding and compiling

1.2 OpenFOAM core classes

1.3 Tools and practice

■ Boundary conditions

4.1 Class hierarchy

4.2 Generic Programming

4.3 Class Functions / Data

4.4 Design and implementation

■ Utility applications

2.1 Data Construction

2.2 Data Access

2.3 Data Input

2.4 Data Operations

■ Solver applications

5.1 Fields

5.2 Equation and algorithms

5.3 Derivatives and algebra

■ Model development

3.1 Object Orientation

3.2 Interfaces

■ Data Analysis

6.1 Data Processing

6.2 Data Output

- Task: create an application called `cppSandbox` that:
 - ...creates an integer $a = 2$ and real number $b = 3.4$; calculates $a \times b$;
 - ...and writes the answer to standard output (the terminal)

- From the `run` directory, make a `cppSandbox` directory; go into it

```
>> run  
>> mkdir cppSandbox  
>> cd cppSandbox
```

- Open an editor, e.g. gedit, and create a file named `cppSandbox.C`

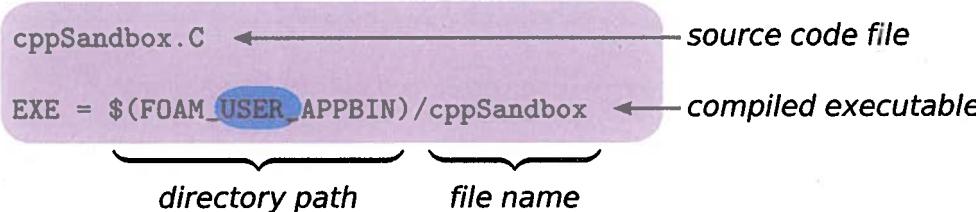
```
#include <iostream>

int main()
{
    int a = 2;
    double b = 3.4;

    std::cout << a << " * " << b << " = " << a*b << "\n";

    return 0;
}
```

- Use wmake compilation tool
- Run wmakeFilesAndOptions: creates Make dir. with files & options files
>> wmakeFilesAndOptions
- Edit files:



- Compile with wmake
>> wmake
- Run the application
>> cppSandbox
 $2 * 3.4 = 6.8$ ← from cout<<...
- Check the return value
>> echo \$?
0 ← from return (0);

- Task: add an additional calculation of $a \times b$ with $a = 2.1$
- Add an inner block to `cppSandbox.C` and recompile

```
#include <iostream>

int main()
{
    int a = 2;
    double b = 3.4; }

    {
        double a = 2.1;
        std::cout << a << " * " << b << " = " << a*b << "\n";
    }

    std::cout << a << " * " << b << " = " << a*b << "\n";

    return 0;
}
```

*a and b names introduced — declaration
... with values specified — definition
within the scope of the main function block — { ... }*

*a redefined within inner block
b used from outer block*

- a, b are objects
- All objects have type: `int = integer; double = double precision number`
- Run the `cppSandbox` application

- Large software → large amount of code outside scope of `main` function
- Lots of non-local names ⇒ risk of name conflicts
- `namespace` = named scope, e.g.

```
namespace myNS
{
    int a;
}

int main()
{
    myNS::a = 2;
    ...
}
```

names used normally within scope

*names use namespace qualifier
:: when used outside of scope*

- C++ standard library → `std` namespace, e.g. `std::cout<<`
- `using` directive avoids qualifiers

```
using namespace std; // insert directive for std namespace

int main()
{
    ...
    cout<< ...
}
```

*remove std:: qualifier
on cout<< functions*

- Task: add a cout<< statement that writes b^2

- Before return (0); line, insert

```
cout << "sqr(" << b << ") = " << pow(b, 2) << "\n"
```

- Compiler does not recognise the name of the pow function

```
cppSandbox.C:16:45: error: 'pow' was not declared in this scope  
    cout << "sqr(" << b << ") = " << pow(b, 2) << "\n";
```

- Before int main(), #include the header file containing declaration of pow

```
#include <math.h> ← Header (.h) file for C numerics library
```

- Header files: mainly declarations so compilers recognise names, e.g. of functions

- Compile and run

- Application → source code with `main` function, compiled
- Namespace → named scope of a software `package`, e.g. `std::`, `Foam::`
- Library → module of functionality within package
- Header file → `interface`, e.g. functions, for functionality within library
- Declaration → `type` and `name`, e.g. `int a;`
- Definition → storage additionally allocated, e.g. `a = 2;`
- Online resources:
`LearnCpp.com: http://www.learncpp.com`

eckert
O'Reilly:

- We have used C++ types / functions, e.g. int, double, cout<<
- OpenFOAM contains ~3,000 classes (defined types) with ~50,000 functions
- ...beginning with enhanced versions of C++ primitive types
- Task: create new foamSandbox application using OpenFOAM types

OpenFOAM header file → #include "IOstreams.H"
for streaming I/O

OpenFOAM namespace → using namespace Foam;

int → label
length selectable

double → scalar
precision selectable

cout → Info
handles parallel running

```
int main()
{
    label a = 2;
    scalar b = 3.4;                                means "end line"
    Info << a << " * " << b << " = " << a*b << endl;
    Info << "sqr(" << b << ") = " << sqr(b) << endl;
    return 0;
}
```

- Compile and run
- Names convey meaning: “cell label”; “scalar value”; “Information”

OpenFOAM core classes | scalar, vector, tensor

- Classes for tensors in 3D Cartesian space
- e.g. scalar b , vector $\mathbf{u} = u_i = (u_1, u_2, u_3)$, tensor $\mathbf{T} = T_{ij}$
- Task: instantiate $\mathbf{u} = (1, 2, 3)$, $\mathbf{v} = (2, 3, 1)$ and \mathbf{T}

```
vector u(1, 2, 3);
vector v(2, 3, 1);
tensor T(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- Need `vector.H` to declare `vector`; and `tensor.H`...
- `fieldTypes.H` includes both files, and more

```
#include "fieldTypes.H" ← $FOAM_SRC/OpenFOAM/lnInclude/fieldTypes.H
```

- Add some algebraic operations: \times , $+$, \bullet (inner product)

<code>Info << "b*u = " << b*u << endl;</code>	$= b \times \mathbf{u}$
<code>Info << "u + v = " << u + v << endl;</code>	$= \mathbf{u} + \mathbf{v}$
<code>Info << "u & v = " << (u & v) << endl;</code>	$= \mathbf{u} \bullet \mathbf{v}$
<code>Info << "T & v = " << (T & v) << endl;</code>	$= \mathbf{T} \bullet \mathbf{v}$

& precedes a \ll ; brackets.

- Compile and run

- Task: create a list of Fibonacci numbers 1, 1, 2, 3, 5, 8, 13, 21, ... $\sim 10^6$
- Create a new `fibonacci` directory and change into it
- Create a `fibonacci.C` file

```
#include "IOstreams.H"
#include "List.H"

using namespace Foam;

List<Type> template
<Type> = any type
e.g. scalar, vector, List

Loop until  $a_{n-1} < 10^6$ 
Appends new value
 $a_n = a_{n-1} + a_{n-2}$ 

int main()
{
    List<label> a(2, 1);

    while (a.last() < 1e6)
    {
        a.append(a.last() + a[a.size() - 2]);
        Info << a << endl;
    }

    return 0;
}
```

- Create Make/files and Make/options, compile and run
- What happens if we increase 10^6 ? Replace label \rightarrow scalar?

comes with github

- Task: obtain list of $2 \times$ Fibonacci numbers, i.e. 2, 2, 4, 6, 10, 16, 26, 42, ...
- Before Info<<: loop over List, doubling each element

```
forAll(a, i) ← forAll macro function: shorthand for
{
    a[i] *= 2;
}
```

- Compile and run → works, but there is a better alternative...
- Field<Type>: derived from List class, but with additional field algebra *emphasis
here!*
- ⇒ replace List<Type> → Field<Type> (including List.H → Field.H)
- Compiles and runs ⇒ Field inherits List functionality, behaviour
- Use field algebra: delete the forAll loop and change Info<<

```
Info<< 2*a << endl;
```

- We need to handle text in programming
- C++ has classes **char** (character) and **string**, e.g. "hello"
- OpenFOAM has **word** class → string without _ / \ ; { } []
- ... good input/output handling
- Add a simple word and include in Info statement

```
word mathematician = "Fibonacci";  
  
Info<< "The series of 2x " << mathematician  
      << " numbers is:" << nl << 2*a << endl;
```

inserts newline character "\n"

- Tensor classes → scalar, vector, tensor, symmTensor, sphericalTensor
- Tensor algebra → $b*v$, $u + v$, $u \& v$, $T \& v$
- List<Type> → template (class)
- forAll → loop over List
- Field<Type> → List with field algebra
- Field is derived class of List base class
- Inheritance → Field has functionality/behaviour of List
- Text → string base class, word derived class

- We have used OpenFOAM classes and functions — where do they come from?
- Task: learn to find classes, functions, etc.
- Programmer's "Doxygen" Documentation — <http://cpp.openfoam.org>
- Open in browser: search for "Field"

Field

Field<Type> a(2, 1); → constructor

Public Member Functions

Field ()
Construct null. More...

Field (const label)
Construct given size. More...

Field (const label, const Type &)
Construct given size and initial value. More...

▶ Public Member Functions inherited from List< Type >

Type & **last ()**
Return the last element of the list. More...

a.last() → Type

▶ Public Member Functions inherited from UList< Type >

void **append (const Type &)**
Append an element at the end of the list. More...

a.append(...Type...)

label **size () const**
Return the number of elements in the UList. More...

a.size() → label

- Very important for managing complexity
- Self-describing code → good naming and code structure — not comments
- `typedef` simplifies names — especially with template classes
- e.g. `Field<scalar>` is clumsy for “scalar field”
- `typedef` searching not effective in Doxygen; command line is better

```
>> find $FOAM_SRC -type f | xargs grep -E 'typedef *Field<scalar>'  
      find in src dir    all files    →  lines with typedef...Field<scalar>  
                                         -E = regular expressions; \* = 0+ spaces
```

- Modify fibonacci to make it easier to read

```
#include "IOstreams.H"  
#include "Field.H"  
#include "primitiveFields.H" ← scalarField.H, vectorField.H, etc. all  
using namespace Foam;  
  
int main()  
{  
    labelField a(2, 1);  
    ...
```

included from primitiveFields.H — reduces the number of header files

- Standard style is essential for avoiding errors, easier analysis, maintainability

- OpenFOAM code style guide: <http://openfoam.org/dev/coding-style-guide>

- In particular, code indentation avoids errors

- OpenFOAM uses Allman-style indentation with 4 spaces, NOT Tabs

https://en.wikipedia.org/wiki/Indent_style#Allman_style

- For example, a conditional statement

```
if (condition)
{
    code;
}
```

- Also, variable assignment and function handling

```
variableName =
    longClassName::longFunctionName
(
    longArgument1,
    longArgument2
);
```

- Max 80 character lines; avoid trailing whitespace

- a + b, a - b, a*b, a/b

- Requires good, configured text editor, e.g. gedit, emacs, atom, vim
- Open Preferences in gedit
Some provided by gedit-plugins

Standard max line length

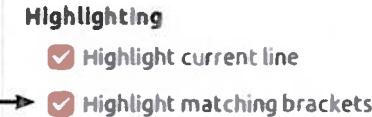
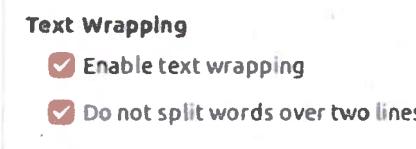


Brackets are aligned

Sub-levels indented 4 spaces

...not Tabs

```
while (a.last() < 1e6)
{
    a.append(...);
}
```



- OpenFOAM contains scripts to create new code: `foamNew...`
- Provide a consistent framework for development
- Task: set up new application called `setBubble` — to be used later
- Run `foamNewApp` to create `setBubble`; compile the code

```
>> run  
>> foamNewApp setBubble  
>> wmake setBubble
```

- View `setBubble.C`

```
#include "fvCFD.H"  
  
int main(int argc, char *argv[]){  
    #include "setRootCase.H" // creates argList named args  
    #include "createTime.H" // checks args against valid args/options  
  
    Info<< nl << "ExecutionTime = " << runTime.elapsedCpuTime() << " s"  
        << " ClockTime = " << runTime.elapsedClockTime() << " s"  
        << nl << endl;  
  
    Info<< "End\n" << endl;  
    return 0;  
}
```

single header file that includes header files for common classes

arguments passed to program e.g. setBubble -help

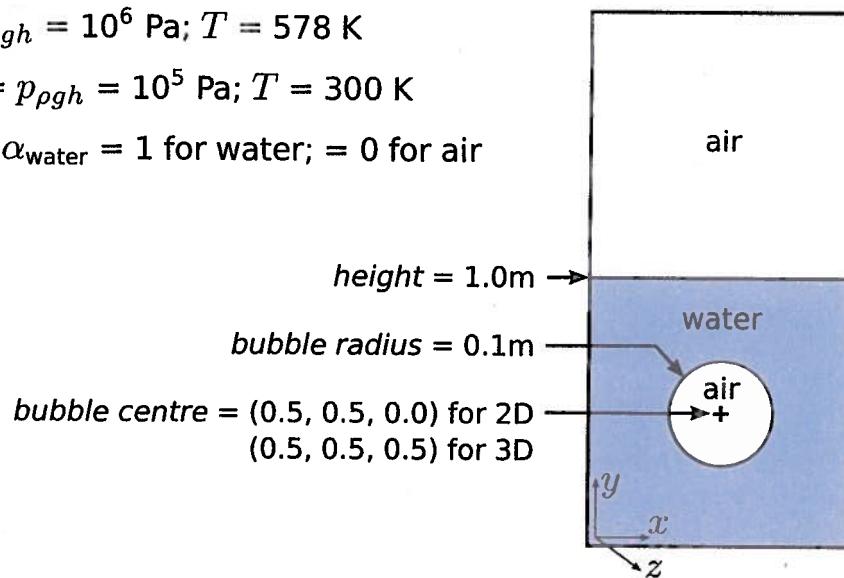
creates argList named args

checks args against valid args/options

- Communication is essential for code maintainability
- Files should be documented in the header
- The Description section provides space for this
- Complete the Description after the code is written

- Doxygen documentation → lists of classes & functions
- ...especially good for providing inherited functions
- Code readability → good naming, use of `typedef`
- Programming style → indentation reduces errors, consistency aids analysis
- Good text editor → configure to support style
- `foamNew...` scripts → good templates that conform to style

- Some simulations need non-uniform initialisation of fields, e.g. p, T
 - e.g. `depthCharge2D` and ... 3D tutorial cases for `compressibleInterFoam`
 - Task: program `setBubble` to initialise an air bubble and air/water interface
-
- Bubble: $p = p_{\rho gh} = 10^6 \text{ Pa}$; $T = 578 \text{ K}$
 - Elsewhere: $p = p_{\rho gh} = 10^5 \text{ Pa}$; $T = 300 \text{ K}$
 - Phase fraction $\alpha_{\text{water}} = 1$ for water; = 0 for air



- Time class: database (objectRegistry) and time data (e.g. time step)
- Code template includes `createTime.H`, a short piece of code — in effect:

`Time runTime("controlDict", args);` ← just a heavily reused piece of code
NOT a class declaration (header) file

class object read file case info

- After Time is created, data is read from file, starting with mesh, then fields
- fvMesh class: mesh data and functions, for finite volume calculations
- Conveniently added by including `createMesh.H`; add at line 42

42 `#include "createMesh.H"`

line number

class object

```
fvMesh mesh
(
    IOobject
    (
        "region0",
        runTime.timeName(),
        runTime,
        IOobject::MUST_READ
    )
);
```

name of registered object → "region0",
directory data is read from → runTime.timeName(),
object registry → runTime,
data must be read in → IOobject::MUST_READ

- GeometricField template class: field (e.g. p , T), including BCs, dimensions
- Create a local `createFields.H` file and include it in `setBubble.C`

43 #include "createFields.H"

- We need field creation code \Rightarrow copy an existing `createFields.H` file, e.g.
`>> cp $FOAM_UTILITIES/preProcessing/boxTurb/createFields.H .`
- Modify `createFields.H` to read in pressure field p

```
Info<< "Reading field p\n" << endl;
typedef for GeometricField of → volScalarField p
scalars defined at cell centres (vol)
(
    IOobject
    (
        e.g. reads from { file/object name → 'p' ,
        O/p file   time directory → runTime.timeName() ,
        object registry → mesh,
        must read data → IOobject::MUST_READ,
        "automatic" writing → IOobject::AUTO_WRITE
    ),
    mesh
);
```

- Compile the application

- Duplicate the entire entry for p 3 times in `createFields.H`
- Replace p with p_rgh (3 occurrences) in the first duplicate, T in the second
- ... and `alpha.water` in the last duplicate, but name the object just alpha

```
Info<< "Reading field alpha.water\n" << endl;
volScalarField alpha
(
    IOobject
    (
        "alpha.water",
        ...
    )
)
```

- Task: initialise alpha field to 1 in all cells by "alpha = ..."
- Fields contain a `dimensionSet`: dimensional units [M L T K Q C L]
- Field algebra does dimension checking on left/right of +, -, =
- ... so generally we need dimensions on right of alpha = ...

- `dimensioned<Type>` class → `<Type>` (scalar, vector, ...) with dimensions
- Data → word, dimensionSet, `<Type>`

```
56 alpha = dimensionedScalar("alpha", dimensionSet(0, 0, 0, 0, 0), 1.0);
```

↑ ↑ ↑
word dimensionSet scalar

- Here, `dimensionedScalar` is constructed "on-the-fly"

```
class1 object1(args);      → object2 = class1(args);  
object2 = object1;
```

- For readability → predefined dimension sets with familiar names

- ... extern (global) variables, e.g. `dimless`, `dimMass`, `dimPressure`

see \$FOAM_SRC/OpenFOAM/dimensionSet/dimensionSets.C

modify

```
56 alpha = dimensionedScalar("alpha", dimless, 1.0);
```

- Compile again

- Construct `centre = (0.5, 0.5, 0)`
- `centre` is fixed: construct with `const` to restrict modifying it (by mistake)

```
58 const point centre(0.5, 0.5, 0);
```

↑
typedef for vector
construct const

- Construct `radius = 0.1, height = 1.0, e.g.`

```
59 const scalar radius = 0.1;
```

↑
generally use assignment (=) for
primitives, instead of "radius(0.1);"

- Construct bubble pressure $pB = 10^6$, $TB = 578$
- Compile again; warnings issued for unused primitives — that's OK

```
createFields.H:59:14: warning: unused variable 'radius' [-Wunused-variable]
  const scalar radius(0.1);
```

- `Time` → object registry and time data
- `fvMesh` → mesh for finite volume calculations
- `vol<Type>Field` → fields with values at cell centres
- `dimensioned<Type>` → scalar, vector, ... with dimensions
- `const` → for any object that will not be modified
- `scalar, label` → construct using `=`, e.g. `"label i = 0;"`
- `typedef` → use to improve understanding

- Classes: defined entities with data and functions
- Object: e.g. `mesh` → construction of a particular class, e.g. `fvMesh`
- Object contains data → how do we access it?
- Task: setting $\alpha_{water} = 0$ inside bubble and above interface
 - ⇒ for every cell, check if centre **C** is inside bubble or above interface
 - ⇒ need **C** data → part of `mesh` → look at `fvMesh` class → Doxygen

can be called from outside class definition, e.g. from an application

Public Member Functions

return type	functions
<code>const DimensionedField< scalar, volMesh > &</code>	<code>V () const</code> Return cell volumes. More...
<code>const surfaceVectorField &</code>	<code>Sf () const</code> Return cell face area vectors. More...
<code>const surfaceScalarField &</code>	<code>magSf () const</code> Return cell face area magnitudes. More...
<code>const volVectorField &</code>	<code>C () const</code> Return cell centres as volVectorField. More...
<code>const surfaceVectorField &</code>	<code>Cf () const</code> Return face centres as surfaceVectorField. More...

- C() function: returns cell centres — a set of vectors
- volVectorField: vectors defined at cell centres
- We want to access cell centre data already calculated and stored
- ... with a reference to the data → like a pointer, but not reallocatable, so safer
- ... equivalent to symbolic link or shortcut in a filing system
- Create a const reference, so data modification is restricted

```
45 const volVectorField& C = mesh.C();
```

↑
& means "reference"

- Now start looping over cells to set values of alpha

```
47 forAll(alpha, celli)  
48 {
```

↑
loop index

- Now what? Try writing the code to set alpha in bubble and above interface

- UList elements can be accessed with the array subscript operator []

T & operator[] (const label)

Return element of UList. More...

const T & operator[] (const label) const

Return element of constant UList. More...

*both const and non-const functions available
⇒ [] can be used to access or modify an element*

- ⇒ cell centre values accessed by "C[celli]"; create a reference...

49 const vector& Ci = C[celli];

- Why? 1) Need the data twice → more efficient to call the function once
- ... 2) Ensures the specific **const** version of the function
- ... 3) Improves readability

- Write the code to set alpha

```
47 forAll(alpha, celli)
48 {
49     const vector& Ci = C[celli];
50
51     if (mag(Ci - centre) <= radius)
52     {
53         alpha[celli] = 0;
54     }
55     else if (Ci.y() > height)
56     {
57         alpha[celli] = 0;
58     }
59 }
```

*Global function of style
function(arg1, arg2, ...)
mag() calculates magnitude*

*Member access function of style
object.function()
relational operator*

assignment operator

- Global functions: commonly used for maths operations

see \$FOAM_SRC/OpenFOAM/primitives/VectorSpace/VectorSpaceI.H

- Operator functions: http://wikipedia.org/wiki/Operators_in_C_and_C++

- Function overloading: single function/operator name, defined multiple times

- ... operates differently depending on context, e.g. for operator&

$(\text{bool} \ \&\& \ \text{bool}) = \text{AND operation}; (\text{tensor} \ \&\& \ \text{tensor}) = \text{double inner product } \mathbf{T}:\mathbf{T}$

- alpha has been initialised; we now need to write data into 0/alpha.water

```
61 alpha.write(); ← writes to location according to IOobject  
dir: runTime.timeName(); file: "alpha.water"
```

- Compile; test the application on the depthCharge2D case

```
>> run  
>> cp -r $FOAM_TUTORIALS/multiphase/compressibleInterFoam/laminar/depthCharge2D .  
>> cd depthCharge2D  
>> blockMesh  
>> cp -r 0.org 0  
>> setBubble
```

- Check 0/alpha.water in ParaView

- Set p = pB and T = TB in bubble; and p_rgh = p

```
54 p[celli] = pB;      62 p_rgh = p;  
55 T[celli] = TB;
```

- Write out alpha, p, T, p_rgh; instead of 4 write(), replace alpha.write() by

```
64 runTime.writeNow(); ← writes all objects registered to runTime  
with IOobject::AUTO_WRITE option
```

- Access functions → e.g. `mesh.C()`
- Reference → (`&`) points to existing data stored in memory
- `const` declaration → restricts modification of data
- Global functions → e.g. `mag(C)`, often used for better style
- Function overloading → e.g. `alpha.write()` writes a field
- ...`runTime.write()` writes database (more to follow...)

- Input data is currently hard coded → inflexible; bad practice
- Task: modify the code to read data from file
- Case properties are in `constant/...Properties` files — a dictionary
- Plan: read input parameters from `constant/bubbleProperties`
- In `createFields.H`, construct `I0dictionary`: registered dictionary object
- Start at line 1 because nothing can happen without this data
- Copy/paste a sample of similar code, e.g. lines 1-13 of
`$FOAM_SOLVERS/incompressible/icoFoam/createFields.H`

```
Info<< "Reading bubbleProperties\n" << endl;
I0dictionary bubbleProperties
(
    I0object
    (
        file/object name      "bubbleProperties",
        constant directory    runTime.constant(),
        object registry        mesh,
        must read data         I0object::MUST_READ,
        no writing             I0object::NO_WRITE
    )
);
```

- OpenFOAM uses dictionary I/O format with keyword lookup
- Test the lookup of `centre` parameter
- Copy `turbulenceProperties` to create `bubbleProperties` in `depthCharge2D`
- Delete existing entry and add `centre` entry

```
18 centre (0.5 0.5 0); ← note: input data, not code → no commas  
                                data is delimited by spaces
```

- Data is looked up with the `lookup("keyword")` function; add

```
14 const point centre  
15 (  
16     bubbleProperties.lookup("centre")  
17 );
```

→ `centre (0.5 0.5 0);`

← `I0dictionary in memory`

- Remove old constructor of `centre` (line ~75); recompile
- Test again; view in ParaView; move the `centre`

- Task: add code for other properties
- Some properties relate to bubble, others to liquid (e.g. height level)
- To lookup centre, construct a reference to the bubble sub-dictionary

```
14 const dictionary& bubbleDict(bubbleProperties.subDict("bubble"));
```

- Create sub-dictionaries for bubble and liquid in bubbleProperties
- Modify the centre lookup()

```
15 const point centre  
16 {  
17     bubbleDict.lookup("centre")  
18 };
```

- Remove the old centre constructor
- Compile and test

include the liquid properties also {

```
bubble  
{  
    centre (0.5 0.5 0);  
    radius 0.1;  
    p      1e6;  
    T      578;  
}  
  
liquid  
{  
    height 1.0;  
    p      1e5;  
    T      300;  
}
```

- `lookup("chars")` recognises data type based on form of "chars"
- e.g. (0 1 2) is a vector
- Cannot distinguish scalar and label, e.g. "32"
- ... so need type conversion functions `readScalar(...)`, `readLabel(...)`
- Add code to lookup other entries for bubble

```
19 const scalar radius(readScalar(bubbleDict.lookup("radius")));
20 const scalar pB(readScalar(bubbleDict.lookup("p")));
21 const scalar TB(readScalar(bubbleDict.lookup("T")));
```

- Your tasks:
- Remove the redundant constructors for `radius`, `pB`, `TB`
- Create liquid sub-dictionary and lookup `height`, `pL`, `TL` (liquid pressure, temp)
- Assign `pL` to `p` and `p_rgh`; and `TL` to `T`
- Compile and test

- We have the assignment of one GeometricField to another with
`p_rgh = p;`
- This "assigns values of p to p_rgh" in cells
- ... but what happens on boundary patch faces?
- Answer: values are assigned on some BCs, but not on others
- e.g. values are not assigned on `fixedValue`, the BC is unchanged
- ... because the `fixedValue` code has no `operator=` function
- ... but it does have an `operator==` function that sets equivalent values
- ⇒ values on `fixedValue` BCs can be over-ridden by:
`p_rgh == p;`

- Execution parameters can be provided through command line
- `argList` class manages **mandatory** arguments and **options**
- Includes default options, e.g. `-case`; options can be added and removed
- Task: add option to prevent overwriting original fields by incrementing time
- Add a bool option to `argList::validOptions` static data member
- ... before `args` is constructed (`setRootCase.H`)

```
37 argList::addBoolOption  
38 {  
39     "noOverwrite",  
40     "increment time to avoid overwriting initial fields"  
41 };
```

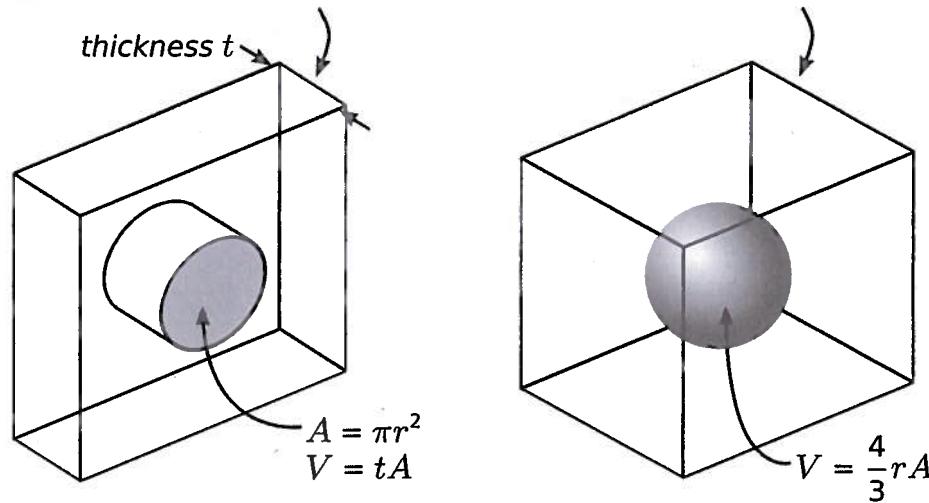
- Increment time before writing data if `noOverwrite` option selected

```
69 if (args.optionFound("noOverwrite"))  
70 {  
71     runTime++;  
72 }
```

- Compile and test

- `I0dictionary` → reads dictionary and registers object
- `lookup()` → looks up data from dictionary
- `subDict()` → references a sub-dictionary
- `readScalar()/readLabel()` → needed for scalar/label lookup
- `dimensioned<Type>` → think “does this assignment need dimensions?”
- `argList` → stores configurable arguments and options

- setBubble utility initialises a bubble by setting $\alpha_{\text{water}} = 0$ in cells
- Task: compare numerical bubble volume with analytical calculation
- Assumption: 2D \rightarrow 'cylindrical' bubble; 3D \rightarrow spherical bubble



- Which class provides cell volumes, 2D/3D information (maybe?), thickness t , etc.?
- Open documentation in Doxygen

- Cell volumes available by the following function, see `fvMesh.H` / page 31

```
294 // - Return cell volumes
295 const DimensionedField<scalar, volMesh>& V() const;
    }  
    return type  
} const function: does not  
modify private data
```

- Create a reference to cell volumes and define `bubbleVolume`

```
51 const DimensionedField<scalar, volMesh>& V = mesh.V();  
52 scalar bubbleVolume = 0.0;
```

Only internal.

- Increment `bubbleVolume` by `V[celli]` for cells within the bubble

```
63 bubbleVolume += V[celli];  
    a += b; → "increase by b"  
    a = a + b; → "take a, add b, assign back to a"
```

- Write `bubbleVolume` to standard output

```
71 Info << "bubbleVolume = " << bubbleVolume << endl;
```

- Compile and test

- Test in parallel on 2 processors with (1 2 1) and (2 1 1) decompositions
- Copy the `depthCharge3D` case locally and use its `decomposeParDict` file
- Run using `mpirun`
`>> mpirun -np 2 setBubble -parallel`
- `bubbleVolume = 0.0065` for (1 2 1); = 0.00325 for (2 1 1) Why?
- sum/min/max functions operate on individual domains
- reduce function → reduction operation across all domain

only shows processor 0
doesn't parallelise well.

```
71 reduce(bubbleVolume, sumOp<scalar>());  
More examples... ↗  
>> find $FOAM_SRC -type f | xargs grep -h "Op<"
```

- Compile and test
- Fields include global operators that embed reductions, e.g. `gSum`, `gMax`, e.g.

```
Info<< "Domain volume = " << gSum(V) << endl;  
see $FOAM_SRC/OpenFOAM/fields/Fields/Field/FieldFunctions.C
```

- Search "dimension" in Doxygen fvMesh to locate functions determining 2D/3D

```
442 // - Return the vector of solved-for directions in mesh.          polyMesh.H
443 const Vector<label>& solutionD() const;
444
445 // - Return the number of valid solved-for dimensions in the mesh
446 label nSolutionD() const;
```

- Implement basic switch statement to calculate input:actual ratio bubble volume

```
73 scalar bubbleRatio =
74     constant::mathematical::pi*sqr(radius)/bubbleVolume; ← = A/VB
75
76 switch (mesh.nSolutionD())
77 { ← constants defined in various namespaces in
    $FOAM_SRC/OpenFOAM/global/constants
78     case 2:
79     {
80         bubbleRatio *= 0.2; // Hard-coded thickness ← = tA/VB
81         break;
82     }
83     case 3:
84     {
85         bubbleRatio *= 4.0*radius/3.0; ← =  $\frac{4}{3}r A/V_B$ 
86         break;
87     }
88 }
```

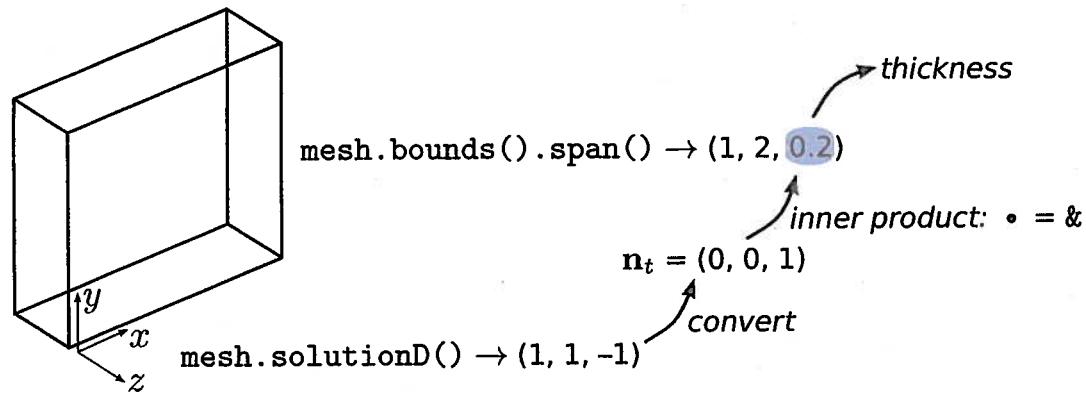
- Always include a default in a switch statement
- FatalErrorInFunction provides location of error in messaging
- ... and causes executable to terminate

```
76 switch (mesh.nSolutionD())
77 {
    ...
88     default:
89     {
90         FatalErrorInFunction
91             << "Case does not appear to be 2D or 3D"
92             << exit(FatalError);
93     }
94 }
```

- Report bubbleRatio and bubbleVolume in Info statement

```
95 Info<< "Specified bubble volume = " << bubbleRatio
96     << " x initialised volume (= " << bubbleVolume << " m^3)" << endl;
```

- Task: calculate thickness instead of hard-coding it
- Tools: `solutionD()` returns a `Vector<label>`
- ... with component 1 for solved direction, -1 for non-solved
- `bounds()` returns `BoundingBox`, then `span()` returns vector (\equiv `Vector<scalar>`)



- Replace `bubbleVolume` scaling for 2D with:

```

80 const vector nT
81 (
82   0.5*vector(Vector<label>::one - mesh.solutionD())
83 );
84 bubbleRatio *= (nT & mesh.bounds().span());

```

Vector<label> type: explicitly converted
to vector, i.e.
Vector<scalar>

Vector is three components.
Vector == Vector<scalar>

- Task: adjust bubble temperature and pressure for error in volume
- ...assuming isentropic expansion; delete assignments to pB, TB, lines 61-62
- In the first loop, build a shortened list of cells for the bubble
- Construct a `DynamicList<label>` named `bubbleCells`

53 `DynamicList<label> bubbleCells;` ← *DynamicList: efficient memory allocation
by doubling its size when needed*

- In the loop, append each bubble cell to `bubbleCells` list

63 `bubbleCells.append(celli);`

- Scale TB, pB assuming isentropic expansion

102 `const scalar gamma = 1.4;`
103 `pB *= Foam::pow(bubbleRatio, gamma);`
104 `TB *= Foam::pow(bubbleRatio, (gamma - 1));`

*namespace
qualifier,
declaration on scalar function*

- Compile and debug
- Write the loop to set p and T in the bubble; and set `p_rgh = p`
- Compile and test

Think in terms of vector algebra.

- `a += b` → compound assignment “increase a by b”
- `reduce(a, sumOp<scalar>())` → reduction operation across parallel domains
- `gSum(a)` → field operation with built in reduction
- switch statement → useful conditional for options
- `FatalErrorInFunction` → error message which causes code to terminate
- `vector(Vector<label>)` → explicit type conversion to vector
- `DynamicList` → efficient extensible list class

- OpenFOAM includes a library of strain-rate dependent viscosity models:
`$FOAM_SRC/transportModels/incompressible/viscosityModels`
- Models are documented in the User Guide: <http://cfd.tips/frhe>
- Task: add the Cross model specified as follows:
https://en.wikipedia.org/wiki/Cross_fluid

$$\nu = \frac{\nu_0}{1 + (k\dot{\gamma})^{1-n}}$$

where: $\dot{\gamma}$ = strain rate; ν_0 , k are model parameters

- Similar to the existing CrossPowerLaw model:
`$FOAM_SRC/transportModels/incompressible/viscosityModels/CrossPowerLaw`
- Each model is specified within its own class (.C and .H files)
- ⇒ We need to create a new class — named **Cross**

- Create a new **Cross** class by 'cloning' **CrossPowerLaw**

- Copy the **CrossPowerLaw** directory, renaming it **Cross**

```
>> run  
>> MODELS=$FOAM_SRC/transportModels/incompressible/viscosityModels  
>> cp -r $MODELS/CrossPowerLaw Cross
```

- Rename **CrossPowerLaw.C** → **Cross.C**; **CrossPowerLaw.H** → **Cross.H**

```
>> cd Cross  
>> rename 's/CrossPowerLaw/Cross/g' *
```

- Replace the word **CrossPowerLaw** → **Cross** inside both files

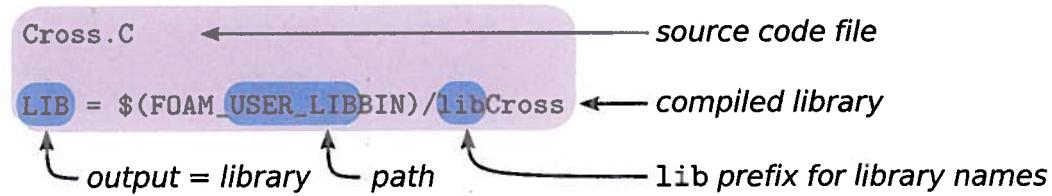
```
>> sed -i 's/CrossPowerLaw/Cross/g' *
```

↑
stream editor
↑ edits files 'in place'
↑ all files
↑
substitute... globallly (= all instances)

- Create **Make/files** and **Make/options** files

```
>> wmakeFilesAndOptions
```

- Edit files:



- Compile the library with wmake → compilation error... why?

```
In file included from Cross.C:26:0:  
Cross.H:38:28: fatal error: viscosityModel.H: No such file or directory  
#include "viscosityModel.H"
```

- Cross class is derived from a base viscosityModel class, see Cross.H

```
53 class Cross ← declares Cross class  
54 :  
55     public viscosityModel ← inherits public members (functions, data)  
56 { ← from viscosityModel class
```

- ⇒ viscosityModel.H must be included in Cross.H (line 38)
- wmake must locate viscosityModel.H to compile
- ⇒ add include directory (-I) and link library (-l) to Make/options

```
EXE_INC = \  
    -I$(LIB_SRC)/finiteVolume/lnInclude \  
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude ← line continuation  
  
LIB_LIBS = \  
    -lfiniteVolume \  
    -lincompressibleTransportModels ← library with compiled  
                                     viscosityModel class  
  
libraries linked to our library ← directory of links  
                               to all .H files in  
                               viscosity models library  
                               incl. viscosityModel.H
```

- Modify model parameters \Rightarrow remove `nuInf`; replace `m` by `k`

- In `Cross.H`:

```
62 dimensionedScalar nuInf_;  
63 dimensionedScalar k_;
```

- In `Cross.C`:

```
100 CrossCoeffs_.lookup("nuInf") >> nuInf_;  
101 CrossCoeffs_.lookup("k") >> k_;  
  
70 nuInf_("nuInf", dimViscosity, CrossCoeffs_),  
71 k_("k", dimTime, CrossCoeffs_),
```

- Modify `calcNu()` function in `Cross.C`:

```
50 Foam::tmp<Foam::volScalarField>  
51 Foam::viscosityModels::Cross::calcNu() const  
52 {  
53     return nu0_ / (scalar(1) + pow(k_*strainRate(), (1 - n_)));  
54 }
```

- `strainRate()` is defined in `viscosityModel.C` \rightarrow `Cross` inherits it

- Test the model on `offsetCylinder` case of `nonNewtonianIcoFoam` solver

```
>> run  
>> cp -r $FOAM_TUTORIALS/incompressible/nonNewtonianIcoFoam/offsetCylinder .  
>> cd offsetCylinder  
>> blockMesh
```

- Select Cross model in `transportProperties`

```
transportModel Cross;  
  
CrossCoeffs  
{  
    nu0      [0 2 -1 0 0 0]  0.01;  
    k        [0 0 1 0 0 0]    2;  
    n        [0 0 0 0 0 0]  0.5;  
}
```

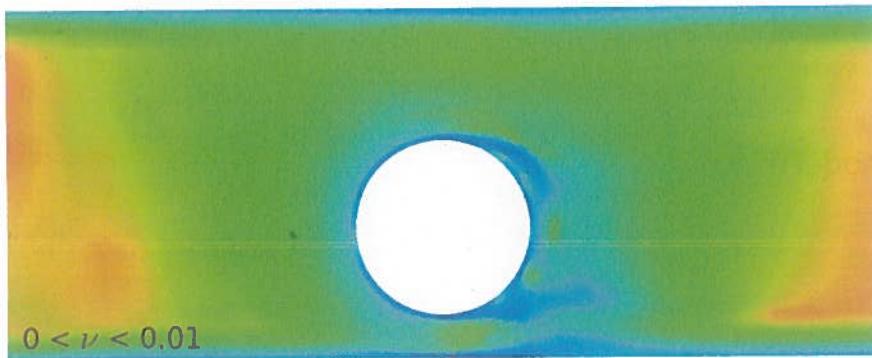
- Load the `libCross.so` library dynamically at run-time by setting in `controlDict`

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

requires full name with .so

- Run the case and view the `nu` field in ParaView

- nu field shows shear-thinning behaviour for $n < 1$



- Try $n = 1.5 \rightarrow$ code blows up in calcNu() function
- $(k\dot{\gamma})^{-0.5} \rightarrow \infty$ (NaN), as $\dot{\gamma} \rightarrow 0$
- \Rightarrow add SMALL (static const scalar) in calcNu()

```
53 return nu0_/(scalar(1) + pow(k_*strainRate() + SMALL, (1 - n_)));
```

see \$FOAM_SRC/OpenFOAM/primitives/Scalar/doubleScalar/doubleScalar.H
\$FOAM_SRC/OpenFOAM/primitives/Scalar/scalar/scalar.H

- Recompile and test

- The Cross model is used when selected in `transportProperties`
- How does the application use the model, compiled in a separate library?
- How does the application get the viscosity from the model?
- The interface is revealed by comparing `nonNewtonianIcoFoam` and `IcoFoam`
- The `meld` application can display the comparison, e.g.

```
>> meld $FOAM_SOLVERS/incompressible/*coFoam/createFields.H
```

```
nonNewtonianIcoFoam/createFields.H      IcoFoam/createFields.H
34 singlePhaseTransportModel fluid        15 dimensionedScalar nu
35 (                                16 (
36   U,                            17   "nu",
37   phi                           18   dimViscosity,
38 );                                19   transportProperties.lookup("nu")
                                         20 );
```

- ⇒ `IcoFoam` constructs `nu` — a single-valued viscosity
- ... `nonNewtonianIcoFoam` constructs `fluid` — a `singlePhaseTransportModel`

- Compare the `nonNewtonianIcoFoam` and `icoFoam` main files

```
>> meld $FOAM_SOLVERS/incompressible/*coFoam/*.C
```

nonNewtonianIcoFoam.C

```
33 #include "singlePhaseTransportModel.H"
```

```
59 fluid.correct();
```

```
67 - fvm::laplacian(fluid.nu(), U)
```

*model updates viscosity
with correct() function*

icoFoam.C

```
64 - fvm::laplacian(nu, U)
```

*model 'delivers' viscosity
with nu() function*

- ⇒ Models are constructed, corrected and delivered

- `singlePhaseTransportModel` is a 'wrapper' around `viscosityModel`

- ... construction of `fluid` → constructs a `viscosityModel` by

```
autoPtr<viscosityModel> fluidPtr ← constructs a pointer  
(                                     to viscosityModel  
    viscosityModel::New("fluid", transportProperties, U, phi)  
);  
viscosityModel& fluid = fluidPtr(); ← 'dereferences' pointer so  
                                         fluid is the viscosityModel
```

- `nonNewtonianIcoFoam` is compiled with `viscosityModel` base class only
- ... but uses `Cross` functions (`nu()`, `correct()`) — through virtual functions
- `viscosityModel` base class includes general features of all viscosity models
- ... including delivery, correction of viscosity → `nu()`, `correct()`

```
viscosityModel.H
148 // Return the strain rate
149 tmp<volScalarField> strainRate() const;
150
151 // Return the laminar viscosity
152 virtual tmp<volScalarField> nu() const = 0;
153
154 // Correct the laminar viscosity
155 virtual void correct() = 0;
```

defined in viscosityModel.C as
 $\sqrt{2.0} \cdot \text{mag}(\text{symm}(\text{fvc}::\text{grad}(\mathbf{U}_)))$

undefined; defined in model
class files e.g. Cross
⇒ declared virtual

- Derived class, e.g. `Cross`, defines virtual functions, e.g. `Cross.H`

```
111 void correct()
112 {
113     nu_ = calcNu();
114 }
```

calcNu() defined in Cross.C

- Compiler points a virtual function to function defined in derived class

- Model is selected at run-time, e.g. through `transportProperties`. How?
- Base class declares/defines a run-time selection 'hash' table
- ... of pointers to constructors, indexed by `typeName`
- Base class has `New` selector function → returns pointer for `typeName`
- Each derived class defines its `typeName` → its model name
- ... and adds its constructor pointer to the table, e.g. `Cross.C`

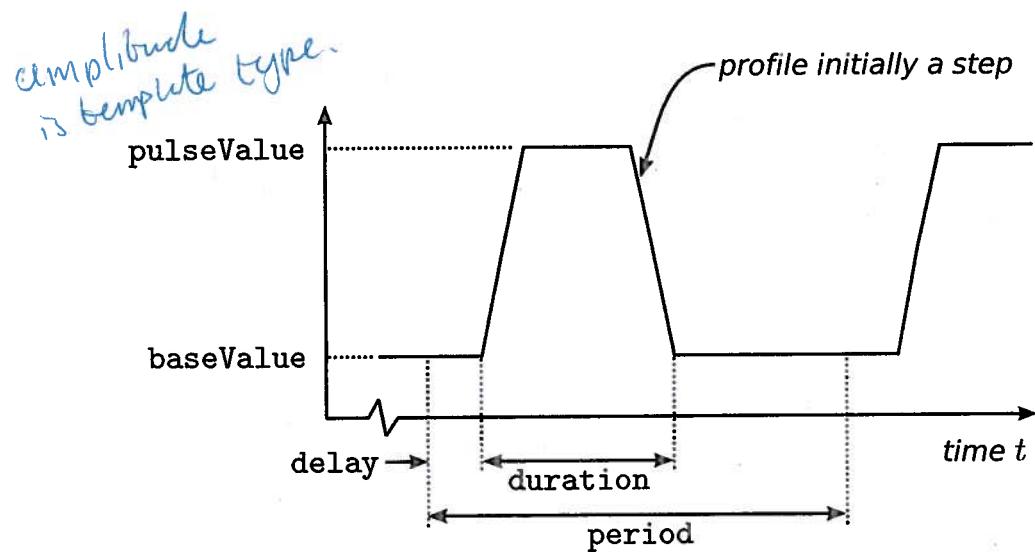
```
36 defineTypeNameAndDebug(Cross, 0);
37
38 addToRunTimeSelectionTable
39 (
40     viscosityModel,
41     Cross,
42     dictionary
43 );
```

*= macro functions defined
in runTimeSelectionTables.H*

- Hash table constructed at run-time
- ⇒ Classes in libraries linked at run-time (through libs) included in hash table

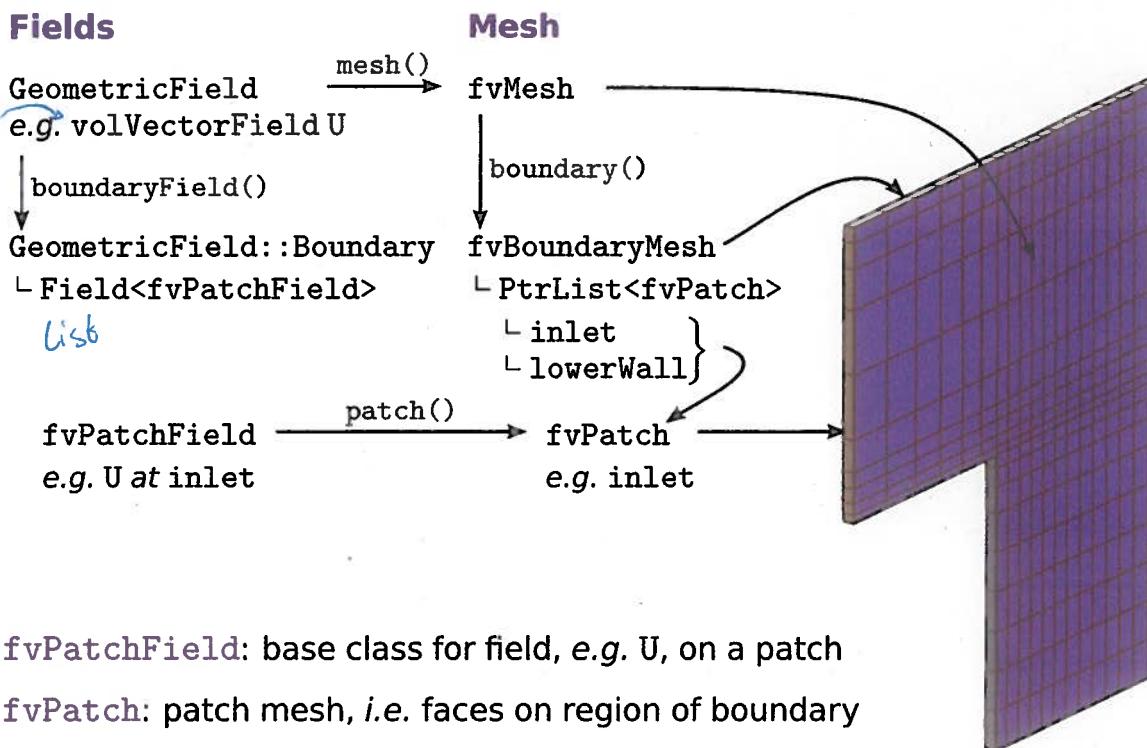
- Models 'plugged into' applications by constructing (pointer to) base class
- Virtual functions defined in base class for use in application
- Virtual functions over-ridden in derived classes
- Application uses functions of the selected model
- Models are selected at run-time through a similar 'virtual' mechanism

- Some inlets are pulsed or time varying, e.g. from a valve
- Task: develop boundary condition (BC) for pulse
- Define the user inputs
 - ... with duration and delay (optional) specified as a fraction of period

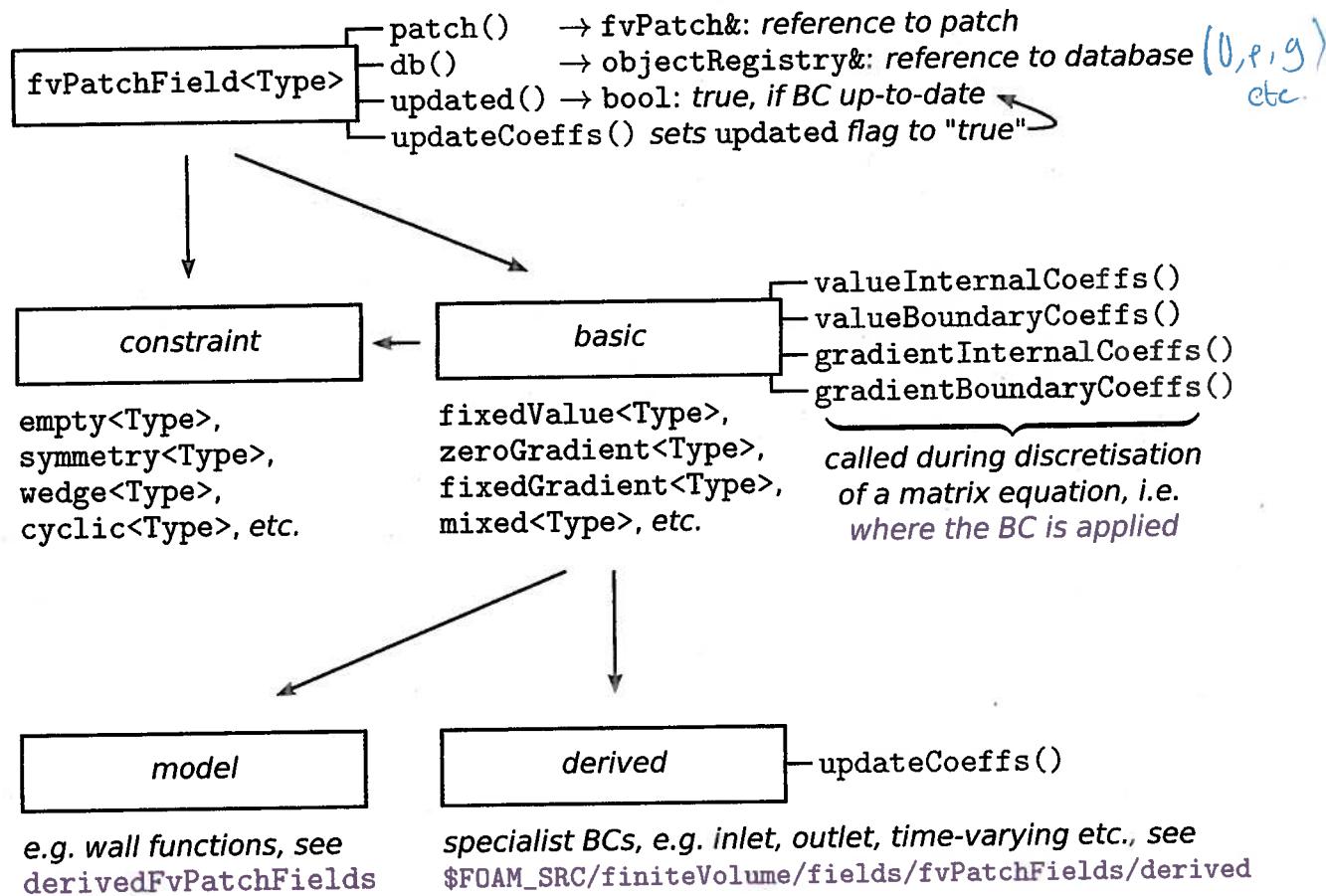


etc /temp^{lates}
snappy stuff

- Consider the mesh and boundary of a case, e.g. inlet section of pitzDaily
- How are BCs stored for fields, e.g. at inlet for velocity U?



- fvPatchField: base class for field, e.g. U, on a patch
- fvPatch: patch mesh, i.e. faces on region of boundary



- New class → unique name
- Derived typically from `fixedValue` (simplest) or `mixed` (advanced)
- BC function, e.g. $\mathbf{Q} = \mathbf{Q}_0(1 + At)$ implemented in `updateCoeffs()`
- Model coefficients, e.g. \mathbf{Q}_0, A → private data
- Case data, e.g. time t → accessed by `patch()`, `db()` functions

- The `foamNewBC` script creates BC class files
- Create the new BC named `pulseFixedValue` as follows

```
>> run  
>> foamNewBC -fixedValue -all pulseFixedValue  
>> cd pulseFixedValue  
>> wmake
```

-`fixedValue` option → derived from `fixedValue`
-`all` option → for all types: scalar (e.g. p), vector (e.g. \mathbf{U}), tensor, etc.

- Compiles into a library named `libpulseFixedValue.so`

- Modify template BC code in `pulseFixedValueFvPatchField.H / .C` files
- Open `pulseFixedValueFvPatchField.H` in an editor
- Lines 99-122 provide common types of private data, e.g. scalar, word, etc.
- Model parameters: period, duration, pulseValue, etc. What types are they?
- First, delete the types we do not need
- Lines 111-122: delete `timeVsData_`, `wordData_`, `labelData_`, `boolData_`

Fields files contain macros to compile
with scalar version, vector version etc.

Type generic for scalar, vector etc.

- Constructor functions → creates an object, initializing data
- ... function name is the same as the class name
- fvPatchFields are constructed when a field, e.g. U, is created
- ... often by reading data from file, e.g. O/U,
- ... but also empty, as a copy, during decomposition, reconstruction, etc.
- ⇒ 5 different constructors defined in pulseFixedValueFvPatchField.C
- Delete initializations of removed private data in **all 5 constructors** starting with...

```
45 pulseFixedValueFvPatchField ← constructor function  
46 ( same name as class  
47     const fvPatch& p,  
48     const DimensionedField<Type, volMesh>& iF  
49 )  
50 :  
51     fixedValueFvPatchField<Type>(p, iF), ← initializations separated  
52     scalarData_(0.0), ← by commas (,) scalarData_ set to 0  
53     data_(Zero),  
54     fieldData_(p.size(), Zero) ← * delete final comma *  
55     timeVsData_(),  
56     wordData_("wordDefault"),  
57     labelData_(-1),  
58     boolData_(false)
```

deconstruct
reconstruct
copy
null
user specified

- Try compiling; compiler reports a lot of error messages, e.g.

```
pulseFixedValueFvPatchField.C:174:21: error: 'timeVsData_' not declared...
    ^file name           ^line number           ^yes, removed from the .H file
```

- Delete removed private data from updateCoeffs() to enable compilation

```
170 fixedValueFvPatchField<Type>::operator==
171 {
172     data_
173     + fieldData_
174     -+ scalarData_*timeVsData_>value(t())
175 };
```

- Similarly, delete removed private data from write() function
- ... referring to compilation error messages for line numbers
- Ensure the code compiles without error

- More code → more maintenance ⇒ avoid code duplication
- What if we have common algorithms/function for different types?
- ... e.g. addition operator+(a, b) for scalar/vector → "add components"
- Use generic programming → templates in C++
- operator+ defined once in VectorSpaceI.H for VectorSpace template class

```
75 template<class Form, class Cmpt, int nCmpt>
76 class VectorSpace
77 { ... }
```

*template arguments, i.e.
types specified later (in <...>)*

- Works for vector class → Cmpt = scalar, nCmpt = 3
- ... tensor class → Cmpt = scalar, nCmpt = 9, sphericalTensor, etc.
- Templates used extensively in OpenFOAM for Type = scalar, vector, tensor + 2
- One set of code for 5 types ⇒ big reduction in duplication

- Many BCs are template classes, e.g. `fixedValue` → applicable to all types
- `pulseFixedValue` is a template class → any field, e.g. `p`, `U` can be "pulsed"

```
93 template<class Type>
94 class pulseFixedValueFvPatchField
95 :
96     public fixedValueFvPatchField<Type> // derived from fixedValue
97 { ... }
```

template argument Type : scalar, vector, tensor, etc.

template class

- Private data, line 106, Type: single scalar, vector, etc.
- Let's use this for `baseValue`, which we can default to zero
- Private data, line 110, `Field<Type>` is "set of values on patch faces, of Type"
- Let's use this for `pulseValue`, allowing different values on each patch face
- Rename `data_` and `fieldData_` in .H file

```
103 // Base value of the field
104 Type baseValue_; // OpenFOAM standard: private data
105 names end with underscore _ to avoid
106 // Value during the pulse ambiguity with function arguments
107 Field<Type> pulseValue_;
```

- Private data requires initialization in constructors
- ⇒ need to rename data and fieldData in .C file, e.g.

```
43 template<class Type>
44 Foam::pulseFixedValueFvPatchField<Type>::
45 pulseFixedValueFvPatchField
46 (
47     const fvPatch& p,
48     const DimensionedField<Type, volMesh>& iF
49 )
50 :
51     fixedValueFvPatchField<Type>(p, iF),
52     scalarData_(0.0),
53     baseValue_(Zero),
54     pulseValue_(p.size(), Zero)
55 {
56 }
```

← template function
← class scope
← function name
} arguments
← initialization follows...
← base class initialized
← scalar initialized to 0
← Type initialized to zero

- How do we initialize zero for Type → scalar, vector, etc. ('0' won't work)
- Use static variable Zero, defined for all primitive types
- Compile, ensuring all data and fieldData entries are updated in .C file

- Constructor from dictionary (lines 59-84) used when a field is read from file
- For example, in O/p, an inlet could use pulseFixedValue by

```
inlet
{
    type      pulseFixedValue;           ← BC typeName
    pulseValue uniform 10;             ← Field<Type> ⇒ uniform/nonuniform
    baseValue  0;                     ← Type ⇒ single value
    ...
}
```

- Default values can be included → used if the keyword entry is omitted
- baseValue currently looked up; requires explicit type conversion to Type

70 baseValue_(pTraits<Type>(dict.lookup("baseValue"))),
 like readScalar, converting Istream (characters) to scalar

- Change lookup to include a default of zero

70 baseValue_(dict.lookupOrDefault<Type>("baseValue", Zero)),
 Type is known ⇒ conversion is built-in

no need for pTraits
 as lookUpOrDefault
 returns a copy

pTraits<Type>::zero is built in
 for arbitrary 0
 or (0, 0, 0)

- Add scalar parameters period and duration by
 - ... 1. replace scalarData with period in .H and .C files
 - ... 2. duplicate period declarations/initializations, renaming duration, e.g.

```
100 //-- Cycle period
101 scalar period_;
102
103 //-- Pulse duration as fraction of cycle period
104 scalar duration_;
```

- Terminate run if input does not satisfy $0 < \text{duration} < 1$; in .C file

```
70     period_(readScalar(dict.lookup("period"))),
71     duration_(readScalar(dict.lookup("duration"))),
72     ...
73 {
74     if (duration_ < 0 || duration_ > 1)
75     {
76         FatalErrorInFunction
77             << "duration = " << duration_ << " is invalid (0 < duration < 1)"
78             << exit(FatalError);
79     }
80 }
```

- BC can be specified as $\alpha \times \text{pulseValue} + (1 - \alpha) \times \text{baseValue}$
- Create `pulseFraction()` function for α
- \Rightarrow Rename the example function `t()` (current time) \rightarrow `pulseFraction()`

```
114 // - Return pulse fraction  
115 scalar pulseFraction() const;  
116  
117 public:
```

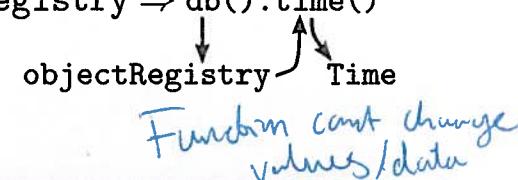
\leftarrow declared before public specifier
 \Rightarrow private (the default) to class code

- `pulseFraction()` is a private function \Rightarrow can only be called from in this class
- \Rightarrow For an object constructed from this class named `obj`, you cannot call:
`obj.pulseFraction()`

This \rightarrow reserved pointer which will point to something you will create used for templates as its going to be created.

class \rightarrow template \rightarrow object

- Define pulseFraction() in .C file
- The function requires time t — already exists in the example t() function
- During a run, time stored by Time object on objectRegistry $\Rightarrow \text{db}().\text{time}()$
- 'db()' \rightarrow "call db() on object of this class"
- If ambiguous \rightarrow use reserved *this pointer



```
34 template<class Type>
35 Foam::scalar Foam::pulseFixedValueFvPatchField<Type>::pulseFraction() const
36 {
37     const scalar t = this->db().time().timeOutputValue();
38
39     scalar cycleFraction = fmod(t/period_, 1.0); ←
40     if (cycleFraction > duration_)
41     {
42         return 0.0;
43     }
44     else
45     {
46         return 1.0;
47     }
48 }
```

this->db():
call db() on object
of *this pointer
fractional part of t/τ

fmod = floating point modulus.

If you want to change in future use db()

- BC expression implemented in updateCoeffs() function
- Called when matrix equation is constructed
- Update mechanism avoids multiple calls for a given matrix equation

```
183 template<class Type>
184 void Foam::pulseFixedValueFvPatchField<Type>::updateCoeffs()
185 {
186     if (this->updated()) ← returns updated_ (true/false)
187     {
188         return; ← *this pointer required because
189     } ← updated() is inherited, class is template
190
191     fixedValueFvPatchField<Type>::operator==
192     (
193         baseValue_* (1.0 - pulseFraction())
194         + pulseValue_*pulseFraction() ← *this pointer not required
195     ); ← pulseFraction() defined in this class
196
197     fixedValueFvPatchField<Type>::updateCoeffs();
198 }
```

assign bc's

↑ sets updated_ = true;

- Compile, eliminating any errors

*sets updated_ to true so
theres no recalculating*

- Test the BC on a basic, transient tutorial case, e.g. `pitzDaily`

- Go to the `run` directory and copy the case

```
>> run  
>> cp -r $FOAM_TUTORIALS/incompressible/pimpleFoam/pitzDaily .
```

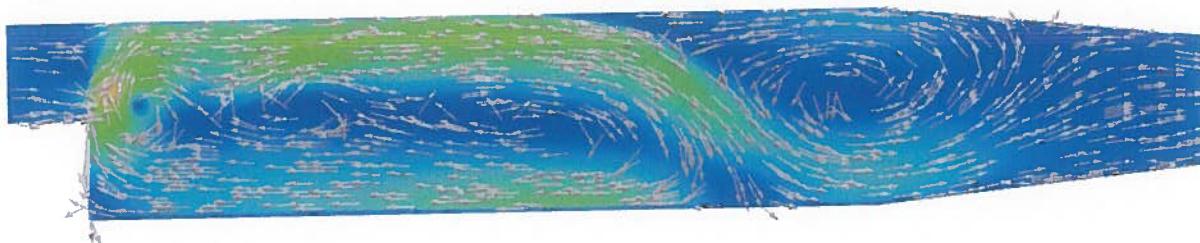
- Edit `controlDict`, linking `libpulseFixedValue.so`, modifying the following

```
libs      ( "libpulseFixedValue.so" );  
endTime  0.1;  
deltaT   0.0001;  
writeControl runTime;  
writeInterval 0.002;
```

- Activate the `pulseFixedValue` BC in `0/U` as follows:

```
inlet  
{  
    type      pulseFixedValue;  ← TypeName: see .H file  
    period    0.01;            ← scalar  
    duration  0.4;            ← scalar: 0→1  
    pulseValue uniform (10 0 0); ← Field<vector>: requires uniform  
    baseValue (1 0 0);        ← vector  
    value     uniform (10 0 0); ← optional, to keep ParaView happy  
}
```

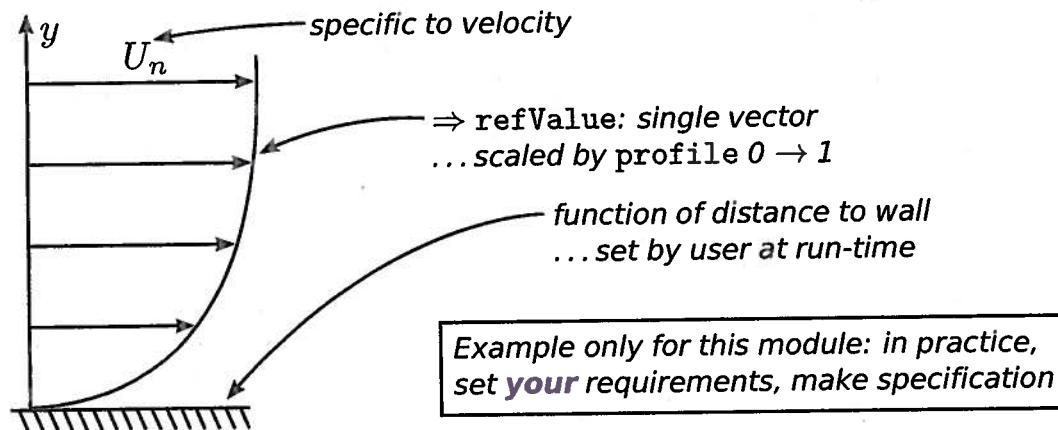
- Run the pimpleFoam solver
- Check results manually, e.g. strip inlet entry from **U** files:
`>> grep -A 8 inlet 0.004/U
>> grep -A 8 inlet 0.008/U` *next 8 lines*
- Examine the simulation in ParaView



- Correct the Description section at the top of the .H file
- Format of Description designed for special typesetting in Doxygen HTML

```
30 Description
31   This boundary condition provides a pulseFixedValue condition,
32   calculated as:
33
34   \f[
35     Q = (f - 1)*baseValue + f*pulseValue
36   \f]
37
38   where
39   f = 1, when fractional part of (time/period) < duration
40   f = 0, otherwise
41
42   \heading Patch usage
43
44   \table
45     Property | Description           | Req'd? | Default
46     period   | cycle period [s]       | yes    |
47     duration | duration, fraction of period | yes    |
48     baseValue | single Type value      | no     | zero
49     pulseValue | Type field across patch | yes    |
50   \endtable
```

- Task: create a velocity BC with a spatial profile
- Requirements?: immediate application? future applications? data input?
- ...usability? options? extensibility? maintainability? efficiency?



- Specification: new `fixedValue`, vector BC → `profileVelocity`
- ...using `Function1` for run-time function; `wallDist` provides distance to wall

- From the `run` directory, create new BC code
`foamNewBC -f -v profileVelocity`
-f → derived from `fixedValue`
-v → for vector field only
- Open `profileVelocityFvPatchVectorField.H` in an editor
- Model parameters: `data_` → `refValue`; `timeVsData_` → `profile`
- Lines 113-121: delete the types we do not need `wordData_`, `labelData_`,
`boolData_`
- Lines 106-109: delete `fieldData_`
- Lines 99-101: delete `scalarData_`

- Edit all 5 constructors in the .C file, deleting initializations, starting with...

```
43 profileVelocityFvPatchVectorField
44 (
45     const fvPatch& p,
46     const DimensionedField<vector, volMesh>& iF
47 )
48 :
49     fixedValueFvPatchVectorField(p, iF),
50     scalarData_(0.0),
51     data_(Zero),
52     fieldData_(p.size(), Zero),
53     timeVsData_(), ← * delete final comma *
54     wordData_("wordDefault"),
55     labelData_( 1 ),
56     boolData_(false)
```

- Delete private data in updateCoeffs() function to enable compilation

```
156 fixedValueFvPatchVectorField::operator==
157 (
158     data_
159     + timeVsData_->value(t())
160 );
```

- Patch fields can be constructed during decomposition and reconstruction
- Construction of patchField data (one value per face) is complex
- ... generally requiring mapping of values onto new patchField
- The autoMap and rmap functions perform the mapping; delete for fieldData_

```
124 void Foam::profileVelocityFvPatchVectorField::autoMap
125 (
126     const fvPatchFieldMapper& m
127 )
128 {
129     fixedValueFvPatchVectorField::autoMap(m);
130     fieldData_.autoMap(m);
131 }
```



```
140 fixedValueFvPatchVectorField::rmap(ptf, addr);
141
142 const profileVelocityFvPatchVectorField& tiptf =
143     refCast<const profileVelocityFvPatchVectorField>(ptf);
144
145 fieldData_.rmap(tiptf.fieldData_, addr);
```

- Correct the write() function and compile

- The Function1 class provides a run-time function of single dependent variable
- ... often a function of time t ; here distance to wall y
- Rename `timeVsData_` → `profile_`; change to scalar function, e.g. .H file

```
103 // Profile as function of distance to wall  
104 autoPtr<Function1<scalar>> profile_;
```

- Update .C file, renaming to `profile_` and ensuring it is a scalar function
- Rename `data_` → `refValue_`, e.g. .H file

```
99 // Max velocity of the profile  
100 vector refValue_;
```

- Update .C file, renaming to `profile_`
- ... checking the initializations in dictionary constructor

```
65 refValue_(dict.lookup("refValue")),  
66 profile_(Function1<scalar>::New("profile", dict))
```

- wallDist class: distance to the wall, a common requirement in CFD
- ... a MeshObject → constructed once only if needed by New() function

```
static const wallDist& wallDist::New(const fvMesh& mesh)
```

- Distance to the wall accessed from wallDist by y() function

```
const volScalarField& y() const
```

- Task: create a distance to wall function y() on the patch
- ... replacing/reusing the t() function
- Use Doxygen → 'chain' access function, try starting with patch()

1. Get reference to mesh (fvMesh object)
2. Use mesh reference in wallDist::New
3. Get reference to distance to the wall y (volScalarField) from wallDist
4. Return field y() → y on the patch

- Evaluate the profile function in the `updateCoeffs()` function

```
profile_->value(y())
```

- Normalise the profile values (≤ 1)

- Assign the values in the `operator==` function in `updateCoeffs()`

- Compile the library

- Copy the pitzDaily case for simpleFoam, renaming pitzDailyProfile
- Edit controlDict, linking libprofileVelocity.so, modifying the following

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

- Activate the profileVelocity BC in 0/U as follows:

```
inlet
{
    type      profileVelocity;
    refValue  (10 0 0);
    profile   table ( (0 0) (0.008 1) );
    value     uniform (10 0 0);
}
```

profile ramps to 1 at 8mm

- Profile ramps from 0 to 1 at 0.008m from wall
- Run the case and check the profile manually and with ParaView

- Solver application → governing equations, model interface and algorithm code
 - ... to simulate particular physics in CFD
 - Prototype solver → existing solver + additional physics
 - Additional physics later transferred to modelling libraries
-
- Task: develop an application for laminar flow with region of porous media
 - Note: porous media code already exists in fvOptions, custom solvers
 - Modify code from icoFoam, renaming porousIcoFoam

```
>> run  
>> cp -r $FOAM_SOLVERS/incompressible/icoFoam porousIcoFoam  
>> cd porousIcoFoam  
>> mv icoFoam.C porousIcoFoam.C
```

- Edit Make/files:

```
porousIcoFoam.C
```

```
EXE = $(FOAM_USER_APPBIN)/porousIcoFoam
```

compiles in ~/OpenFOAM/ubuntu-4.1/platforms/.../bin

- Run wmake

- createFields.H l 1-20 → viscosity nu lookup() with IOdictionary (see sl. 36)
- line 23 → volScalarField p is constructed with IOobject and fvMesh

```
Info<< "Reading field p\n" << endl;
constructs volScalarField → volScalarField p
(
    arg 1: IOobject → IOobject
    constructed from...
        word → "p",
        fileName → runTime.timeName(),
        objectRegistry → mesh,
        readOption → IOobject::MUST_READ,
        writeOption → IOobject::AUTO_WRITE
    ),
    arg 2: fvMesh → mesh
);
```

typeDef

- What is the volScalarField class?

```
>> find $FOAM_SRC -type f | xargs grep -h 'volScalarField:'  
typedef GeometricField<scalar, fvPatchField, volMesh> volScalarField;
```

actual class name — template with 3 arguments

semi-colon follows
name in typedefs

no semi colon after class declaration → it will
be a typedef.

■ Example pressure p: GeometricField<scalar, fvPatchField, volMesh>

```
GeometricField::Boundary (sub-class)
FieldField<fvPatchField<scalar>>
→PtrList<fvPatchField<scalar>>
```

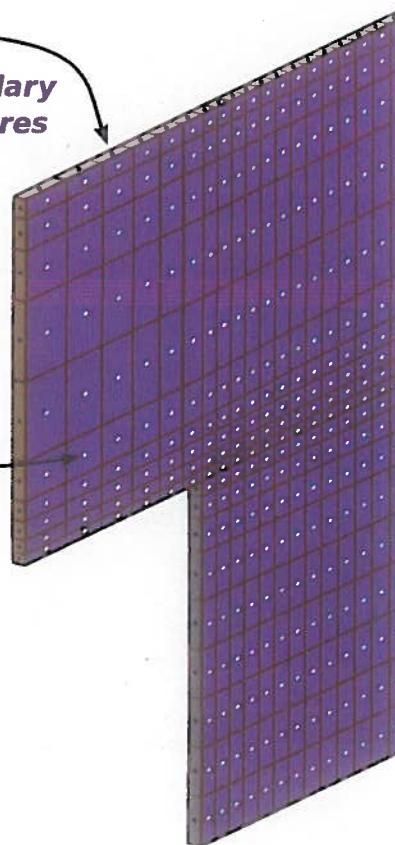
```
boundaryField
{
    inlet
    {
        type zeroGradient;
    }
    ...
}
```

*values at boundary
face centres*

```
GeometricField::Internal (typedef)
```

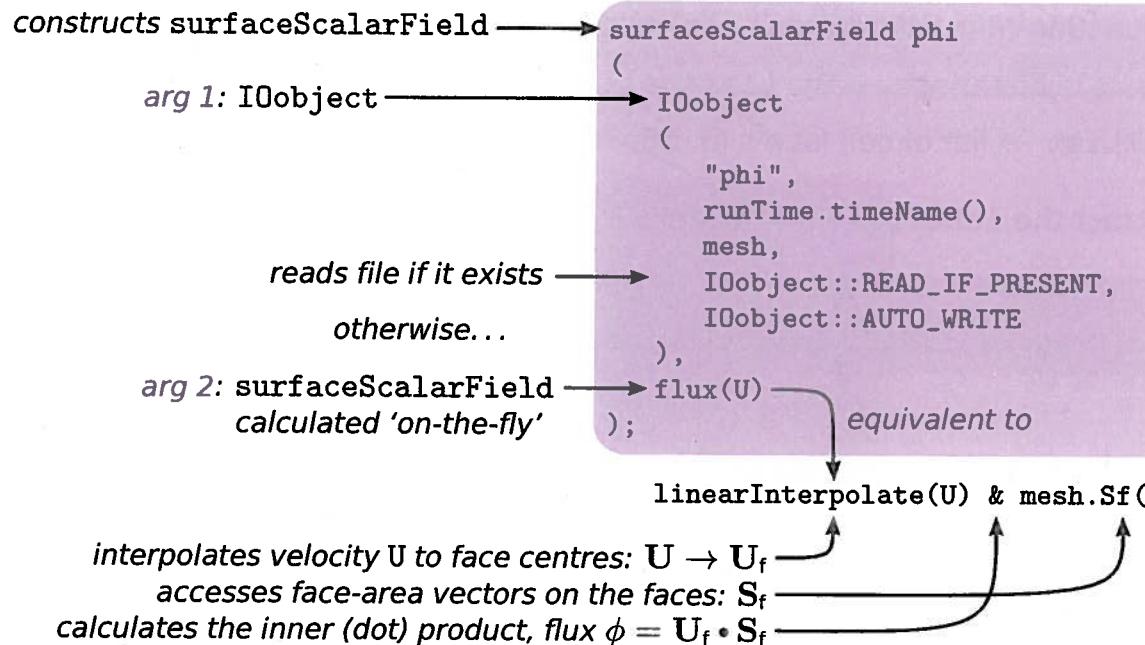
```
dimensions           regIOobject data
[0 2 -2 0 0 0 0];
internalField
nonuniform List<scalar>   IOobject data
2048
(
    0.347229
    0.345293
    ...
)
```

*db()
time()
name()
fvMesh reference mesh()*



- Lines 37-49: volVectorField U constructed → like p, but vector
- Line 52 includes the file `createPhi.H`

`$FOAM_SRC/finiteVolume/cfdTools/incompressible/createPhi.H`



- `surfaceScalarField` → `GeometricField` with values defined at face centres

- Assume Darcy Law, body force $\mathbf{f} = -\nu d \mathbf{U}$, isotropic porosity $d = 1000 \text{ m}^{-2}$
- ... defined in a single cellZone in the mesh
- mesh holds reference to the cell zones → find the access function
- The function (find it!) returns const cellZoneMesh & *-weird templating*
- Show cellZoneMesh → PtrList<cellZone> → PtrList<labelList> *pointer list*
- labelList → list of cell labels (3, 56, 72, ...); PtrList → multiple lists *list of lists*
- Construct the labelList for the cells in the cellZone

```
60 // Porous zone is hard-coded as the first cellZone in the list  
61 const labelList& porousCells(mesh.cellZones()[0]);
```

first element in the PtrList

- Create a field $D = \nu d$, with $d = 1000 \text{ m}^{-2}$ in cells within zone; 0, elsewhere
- Body force → no patch field needed ⇒ use GeometricField::Internal
- Initialise as zero, then set values in cell zone *define as internal field
no need to have boundaries*
- GeometricField::Internal is a typedef for the DimensionedField class

DimensionedField (const IOobject &, const Mesh &mesh, const dimensioned< Type > &, const bool checkIOFlags=true)
Construct from components. More...

```
63 // Darcy momentum coefficient field
64 volScalarField::Internal D
65 (
66     IOobject(
67     (
68         "D",
69         runTime.timeName(),
70         mesh
71     ),
72     mesh,
73     dimensionedScalar("D", dimViscosity/dimArea, 0)
74 );
```

- `readOption` and `writeOption` omitted → default to `NO_READ` and `NO_WRITE`

- Set D values inside the porous zone using `forAll(...)`
- Array operator accesses `value` → need to assign (=) a value
- ... i.e. a scalar, not a `dimensionedScalar`; create the value

```
76 // Darcy constant hard-coded to 1000
77 const scalar dNu = 1000*nu.value();
```

↑
accesses the `value` from
`dimensionedScalar`

- Write the `forAll(...)` loop to set D to dNu in the porous zone
- Compile
- Testing: add Info statement to write D to terminal; recompile
- Copy the `porousBlockage` tutorial (`pisoFoam`) to run directory
 - >> run
 - >> cp -r \$FOAM_TUTORIALS/incompressible/pisoFoam/laminar/porousBlockage .
- Generate mesh, and porous zone with (with `topoSet`), then run the new solver
 - >> blockMesh
 - >> topoSet
- Run the new solver and check D field
 - >> porousIcoFoam | less

- porousIcoFoam.C → ~30 statements (lines ending ';'), 5 controls (if/while)
- Understand the (short) code → modify with confidence
- Starts with (line 32): #include "fvCFD.H" → included single file
- ... containing #include statements for .H files of commonly used classes

```
>> find $FOAM_SRC -name fvCFD.H -type f  
$FOAM_SRC/finiteVolume/cfdTools/general/include/fvCFD.H
```

Has the fvCFD_H token been defined? → #ifndef fvCFD_H
No: define fvCFD_H → #define fvCFD_H
and execute #includes

Yes: omit the file contents → #endif

#include "parRun.H"
#include "Time.H"
#include "fvMesh.H"
... ← header files for Time, fvMesh argList, etc.

- #ifndef...#define...#endif used in .H files to prevent multiple inclusions
- fvCFD.H → makes solver code tidy; convenient, code reuse

- Inside the `main()` function lines 37-51:
 - ... standard `#include` statements to read Time data, mesh,
 - ... fields (`createFields.H`), controls (`pisoControl`, `initContinuityErrors.H`)
- Lines 52-119: solution code, with iterative loops

```
52 while (runTime.loop())  
53 {  
67     if (piso.momentumPredictor())  
68     {  
69         solve(UEqn == -fvc::grad(p));  
70     }  
73     while (piso.correct())  
74     {  
89         while (piso.correctNonOrthogonal())  
90         {  
100            pEqn.solve(mesh.solver(p.select(piso.finalInnerIter())));  
106        }  
110        U = HbyA - rAU*fvc::grad(p);  
112    }  
119 }
```

*increments time (runTime++)
stops if end time exceeded*

user-controlled switch

loops nCorrectors iterations

nNonOrthogonalCorrectors iterations

- `loop()` defined in `Time.C`
- PISO controls (`correct()`, etc.) in `solutionControl.C`, `pimpleControl.C`

- In the momentum equation, add and subtract source term $A\mathbf{U}$

$$\underbrace{\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nu \nabla^2 \mathbf{U} - A\mathbf{U} + A\mathbf{U}}_{-\mathbf{H}(\mathbf{U}), \text{ contains all terms except } \nabla p} = -\nabla p \quad (*)$$

- Divide by A , take divergence and apply $\nabla \cdot \mathbf{U} = 0$

$$-\nabla \cdot \left(\frac{\mathbf{H}(\mathbf{U})}{A} \right) + \cancel{\nabla \cdot \mathbf{U}} = 0 = -\nabla \cdot \frac{1}{A} \nabla p$$

- Pressure equation (the actual one — $1/A$ makes it much more convergent)

$$\nabla \cdot \frac{1}{A} \nabla p = \nabla \cdot \left(\frac{\mathbf{H}(\mathbf{U})}{A} \right)$$

- Momentum corrector, (*) divided by A

$$\mathbf{U} = \frac{\mathbf{H}(\mathbf{U})}{A} - \frac{1}{A} \nabla p$$

- A = diagonal coefficients of momentum matrix equation (PISO, SIMPLE, PIMPLE)

```

60 fvVectorMatrix UEqn
61 (
62     fvm::ddt(U)           -----> [UEqn] → ∂U/∂t + ∇ • UU - ∇ • ν∇U
63 + fvm::div(phi, U)
64 - fvm::laplacian(nu, U)
65 );
66
67 solve(UEqn == -fvc::grad(p)); ----->
68
69 volScalarField rAU(1.0/UEqn.A());
70
71 volVectorField HbyA("HbyA", U);
72
73 HbyA = rAU*UEqn.H();
74
75 surfaceScalarField phiHbyA
76 (fvc::interpolate(HbyA) & mesh.Sf())
77
78
79 fvScalarMatrix pEqn
80 (
81     fvm::laplacian(rAU, p) ==
82     fvc::div(phiHbyA)
83 );
84
85 pEqn.solve(...);
86
87 phi = phiHbyA - pEqn.flux(); -----
88
89 U = HbyA - rAU*fvc::grad(p); -----
90
91 U = HbyA - rAU*fvc::grad(p); -----
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
779
780
781
782
783
784
785
786
787
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
879
880
881
882
883
884
885
886
886
887
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
918
919
919
920
921
922
923
924
925
926
927
927
928
928
929
929
930
931
932
933
934
935
936
937
937
938
938
939
939
940
941
942
943
944
945
946
946
947
947
948
948
949
949
950
951
952
953
954
955
956
957
957
958
958
959
959
960
961
962
963
964
965
966
966
967
967
968
968
969
969
970
971
972
973
974
975
976
976
977
977
978
978
979
979
980
981
982
983
984
985
986
986
987
987
988
988
989
989
990
991
992
993
994
995
996
996
997
997
998
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1805
1806
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1895
1896
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1905
1906
1906
1907
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1915
1915
1916
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1925
1926
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1936
1937
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1946
1947
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1956
1957
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1966
1967
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1976
1977
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1986
1987
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1995
1996
1996
1997
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2005
2006
2006
2007
2007
2008
2008
2009
2009
2010
2011
2012
2013
2014
2015
2015
2016
2016
2017
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2026
2027
2027
2028
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2036
2037
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2046
2047
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2056
2057
2057
2058
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2066
2067
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2076
2077
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2086
2087
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2095
2096
2096
2097
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2105
2106
2106
2107
2107
2108
2108
2109
2109
2110
2111
2112
2113
2114
2115
2115
2116
2116
2117
2117
2118
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2126
2127
2127
2128
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2136
2137
2137
2138
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2146
2147
2147
21
```

- Darcy force $\rightarrow -D\mathbf{U}$ on right hand side (RHS) of '=' in momentum equation
- ... or $+D\mathbf{U}$ on LHS
- Add the term to the `fvMatrix<Type>` class \rightarrow represents matrix equation

fvm:: functions return fvMatrix \rightarrow create M and B

```

61 fvVectorMatrix UEqn
62 (
63     fvm::ddt(U)
64     + fvm::div(phi, U)
65     - fvm::laplacian(nu, U)
66     + D*U
67 );
66 ==
67 - D*U

```

$$\begin{bmatrix} M_{11} & M_{12} & \dots & M_{1N} \\ M_{21} & M_{22} & \dots & M_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ M_{N1} & M_{N2} & \dots & M_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \\ \vdots \\ \mathbf{U}_N \end{bmatrix} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \vdots \\ \mathbf{B}_N \end{bmatrix}$$

$[M] \quad \mathbf{U} = \mathbf{B}$

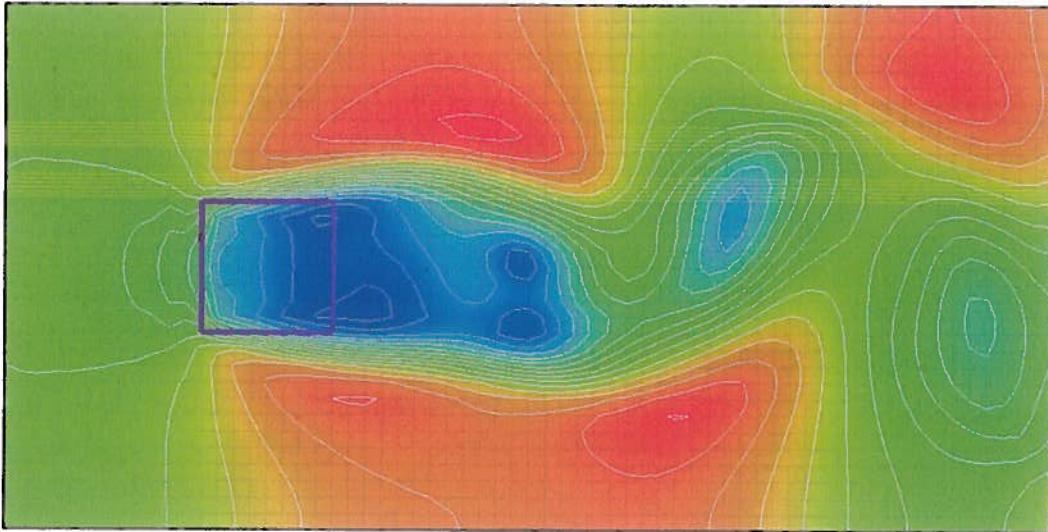
or, alternatively

GeometricField (volVectorField) adds contribution to B

- `fvMatrix<Type>` stores:
 - Matrix coefficients $[M] \rightarrow \text{PtrList<Field<Type>>}$
 - Solution variable $\mathbf{U} \rightarrow \text{GeometricField<Type, ...> \& reference}$
 - Source $\mathbf{B} \rightarrow \text{Field<Type>}$
- Compile the solver



- Run the `porousBlockage` case with `porousIcoFoam`
- View results in ParaView; plot contours $0 < |U| < 1.7 \text{ m/s}$, 0.1 m/s interval



- Porosity creates blockage → flows around blockage, shedding vortices

- Calculations involve derivatives, e.g. $\nabla \cdot$, ∇ , ∇^2 , $\nabla \times$, $\partial/\partial t$
- ... represented by functions, e.g. div, grad, laplacian, curl, ddt
- fvm:: (e.g. fvm::div) → matrix coefficients / source (fvMatrix<Type>)
- fvc:: (e.g. fvc::grad) → calculates new field (vol<Type>Field)
- Fields can provide source to fvMatrix, e.g. D*U, fvc::grad(p)
- Segregated → matrix equation (fvMatrix) solves for one variable
- Decoupled → vectors etc., e.g. U, solves for each component, U_x, U_y, U_z
- ⇒ 4 fvm:: terms → ddt, laplacian, div(phi, ...) (advection), Sp (source)

■ List of derivatives for arbitrary field \mathbf{Q}

Description	Expression	Function
Time derivative [†]	$\partial \rho \mathbf{Q} / \partial t$	fvm::ddt(rho, Q)
Advection [‡]	$\nabla \cdot (\rho \mathbf{U} \mathbf{Q})$	fvm::div(phi, Q)
Laplacian [†]	$\nabla \cdot \Gamma \nabla \mathbf{Q}$	fvm::laplacian(Gamma, Q)
Implicit source	$\rho \mathbf{Q}$	fvm::Sp(rho, Q)
Divergence	$\nabla \cdot \mathbf{Q}$	fvc::div(Q)
Gradient	$\nabla \mathbf{Q}$	fvc::grad(Q)
Curl	$\nabla \times \mathbf{Q}$	fvc::curl(Q)
Explicit source	\mathbf{Q}	Q

[†] Equivalent functions exist for $\partial \mathbf{Q} / \partial t$ and $\nabla^2 \mathbf{Q}$ (without ρ, Γ)

[‡] Advection: fvm::div function with advective flux field phi as the 1st argument

- Equation with time derivative and source

$$\frac{\partial \mathbf{Q}}{\partial t} = a\mathbf{Q}$$

- Discretized $\rightarrow \Delta t = \text{time step}$; 'o' = old time step values

$$(\mathbf{Q} - \mathbf{Q}^o)/\Delta t = a\mathbf{Q}$$

- For $a = 1/\Delta t$, if source is explicit, i.e. uses known values \mathbf{Q}^o

$$\mathbf{Q} - \mathbf{Q}^o = \mathbf{Q}^o \Rightarrow \mathbf{Q} = 2\mathbf{Q}^o$$

- ... but if source is implicit, i.e. the solution variable \mathbf{Q}

$$\mathbf{Q} - \mathbf{Q}^o = \mathbf{Q} \Rightarrow \mathbf{0} = \mathbf{Q}^o \text{ !?}$$

- \Rightarrow Positive sources (right hand side of =) must be explicit
- \Rightarrow Negative sources can be implicit \rightarrow preferred, more stable
- Task: implement Darcy term as implicit source, recompile, test

SuSp Does it for you.

- Isotropic porosity model $\rightarrow \mathbf{f} = -\nu d \mathbf{U}$ with scalar $d = 1000 \text{ m}^{-2}$
- Anisotropic porosity model $\rightarrow \mathbf{f} = -\mu \mathbf{D} \cdot \mathbf{U}$ with tensor \mathbf{D}
- Task: implement anisotropic model with $d_x, d_z = 1000, d_y = 1$ (low resistance in y)

```
63 volTensorField::Internal D
64 (
65     IOobject
66     (
67         "D",
68         runTime.timeName(),
69         mesh
70     ),
71     mesh,
72     dimensionedTensor("D", dimViscosity/dimArea, Zero)
73 );
```

- Change dNu to tensor

```
76 const tensor dNu =
77     tensor(1000, 0, 0, 0, 1, 0, 0, 0, 1000)*nu.value();
```

if anisotropic, must be tensor

■ List of algebraic operations and code functions

Operator	Ranks	Expression	Code
Inner product	≥ 1	$a \cdot b$	$a \& b$
Double inner product	2	$a : b$	$a \&& b$
Cross product	1	$a \times b$	$a \wedge b$
Outer product		$a b \dagger$	$a * b$
Square		$a^2 \equiv aa$	<code>sqr(a)</code>
Magnitude squared		$ a ^2 \equiv a^R \cdot a$	<code>magSqr(a)</code>
Magnitude		$ a \equiv \sqrt{a^R \cdot a}$	<code>mag(a)</code>
Transpose	2	a^T	<code>a.T()</code>
Trace	2	$\text{tr } a = I : a$	<code>tr(a)</code>
Symmetric	2	$\text{symm } a = (a + a^T)/2$	<code>symm(a)</code>
Skew	2	$\text{skew } a = (a - a^T)/2$	<code>skew(a)</code>
Deviatoric	2	$\text{dev } a = a - (\text{tr } a)I/3$	<code>dev(a)</code>
Deviatoric (II)	2	$\text{dev}_{II} a = a - 2(\text{tr } a)I/3$	<code>dev2(a)</code>

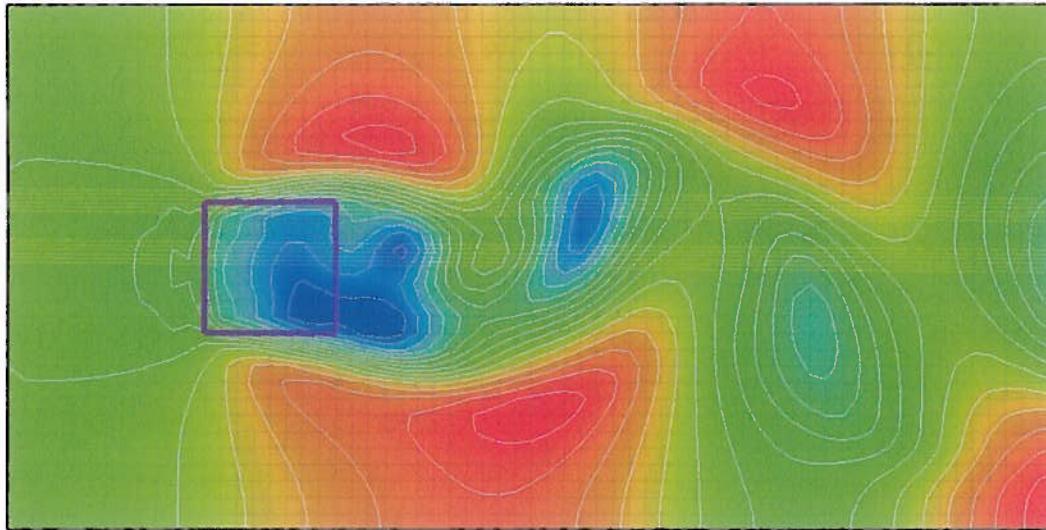
$ta \otimes b$ 'Rank' R: 0 = scalar; 1 = vector; 2 = tensor

■ Modify `porousIcoFoam.C` for tensor D; compile and test

66 + (D & U)



- View results in ParaView; plot contours $0 < |\mathbf{U}| < 1.7 \text{ m/s}$, 0.1 m/s interval



- Challenge: how can the model be implemented in a more implicit way?
- Hint: $\mathbf{D} \cdot \mathbf{U} \equiv [(1/3) \operatorname{tr} \mathbf{D}] \mathbf{U} + \operatorname{dev}(\mathbf{D}) \cdot \mathbf{U}$

- Functionality specified by the user through `functions` in `controlDict` file
- Generally post-processing, calculating time-data and writing to a single file
- ... or larger data sets, e.g. field data, images, written to individual files
- Task: write a function object that writes time-data to file for kinetic energy, K

$$K = \frac{1}{2}m|\mathbf{U}|^2 = \frac{1}{2} \sum_{i=0}^{N-1} \rho_i V_i |\mathbf{U}_i|^2$$

where N = number of cells, V = cell volumes

- Assume initially for incompressible solver, with constant density ρ

postProcess - C++

- The foamNew... script collection includes a script for a new function object

- Create a new function object named `energy`

```
>> run  
>> foamNewFunctionObject energy
```

- Go into the directory and run a test compilation

```
>> cd energy ; wmake
```

- Parameters of the model → private data

$$K = \frac{1}{2} \rho \sum V |\mathbf{U}|^2$$



scalar accessed from object registry, identified by name (word)

- Modify private data in `energy.H`

```
81 // - Name of velocity field on object registry  
82 word UName_;  
83  
84 // - Density  
85 scalar rho_;  
86  
87 // - label  
88 label labelData_;
```

- Modify private data at construction in energy.C

```
53 UName_(dict.lookupOrDefault<word>("UName", "U")),
54 rho_(readScalar(dict.lookup("rho")));
55 labelData_(readLabel(dict.lookup("labelData")))
```

- Modify read() function in energy.C

```
70 dict.readIfPresent("UName", UName_);
71 dict.lookup("rho") >> rho_;
72 dict.lookup("labelData") >> labelData_;
```

- Function objects are called indirectly by Time::loop() function in solver

```
while (runTime.loop())
    ↗
    bool running = run(); ↘
    ↗
    Time.C: functionObjects_.execute(); ↘

    ↗
    FunctionObjectList.C:
        forAll(*this, objectI) ← loops over function objects
        {
            ok = operator[](objectI).execute() && ok;
            ok = operator[](objectI).write() && ok;
        }
    ↗
    ↗ calls execute() and write() ↘
    ↗
    ↗ hold value in execute e.g average. ↘
    ↗
    ↗ writing directly. ↘
```

- Calculate kinetic energy in the `write()` function
- First, access `U` field on the object registry with the `lookupObject()` function

```
91 void Foam::energy::write()
92 {
93     const volVectorField& U = obr_.lookupObject<volVectorField>(UName_);
```

Template function.

base class ↑ ↑ *by default → "U"*

- Include `volFields.H` so the compiler recognises "volVectorField"
- How do we calculate KE — a single value — for the whole domain?
`0.5*rho_*magSqr(U)` calculates one value per cell
- Clue: look at the "finite volume calculate" (fvc) functionality
see `$FOAM_SRC/finiteVolume/finiteVolume/fvc`

```
94     Info<< "Kinetic Energy = "
95         << 0.5*rho_*fvc::...chosen function...(magSqr(U)) << endl;
```

- Include `fvc.H` header file and compile

U · U
magSqr
Volume Integrate

- Test on the pitzDaily case; copy the case to the run directory
- Create a file in system named energy

```
1 // Description
2 //   Writes kinetic energy of entire domain
3
4 type    energy;
5 libs   ("libenergyFunctionObject.so");
6 rho     1.2;
```

- Include the function inside functions in controlDict

```
48 functions
49 {
50     #includeFunc   energy;
51     ...
```

- Run the pitzDaily case; monitor the function object output, e.g.

```
>> simpleFoam | grep "Kinetic"
```

- Output includes a "name" and dimensions. Why?

- Task: modify the code to remove the name and dimensions

Locate List Times -rm

- Document the new function object in the .H file header, e.g.

```
29 Description
30   This function object calculates kinetic energy over the entire domain
31   and writes data to a file
32
33 Example of function object specification:
34 \verbatim
35 energy1
36 {
37   type          energy;
38   functionObjectLibs ("libenergyFunctionObject.so");
39   ...
40   UName        U;      // optional
41   rho          1.205;
42 }
43 \endverbatim
44
45 \heading Function object usage
46 \table
47   Property    | Description           | Required | Default value
48   type        | type name: energy     | yes      |
49   UName       | Name of U field       | no       | U
50   rho         | Density               | yes      |
51 \endtable
```

- Task: modify the function object to write to file, e.g. for graph plotting
- The output to file class, `OFstream`, provides convenient functionality
- ... supported by the `fileName` class
- To set up writing to file, construct an `OFstream` from a `fileName`
- ⇒ create an `OFstream` named `file_` as private data; in `energy.H`

```
87 // - Output file stream  
88 OFstream file_;
```

- In constructor in `energy.C`, initialize `os_` from `fileName`
- ... where the file name is `<functionObjectName>.dat` in the case directory

```
57 rho_(readScalar(dict.lookup("rho"))),  
58 file_(obr_.time().path()/name + ".dat")
```

case directory file name extension

- Compile, including any required header files

- Data needs to be written as a time and value on each line
- ⇒ stream to `file_` instead of `Info` in `write()` function

```
96 file_ << obr_.time()... time access function... << " "
97     << 0.5*rho_*fvc::domainIntegrate(magSqr(U)).value() << endl;
```

- Add a file header comment in the constructor function

```
59 file_ << "# time, kineticEnergy" << endl;
```

- Compile and test
- Column width varies in output file; use TAB spaces
see *punctuation tokens* in `$FOAM_SRC/OpenFOAM/lnInclude/token.H`

```
59 file_ << "# time" << token::TAB << "kineticEnergy" << endl;
```

```
96 file_ << obr_.time().timeName() << token::TAB
```

- Compile and check the output is formatted neatly

- Decompose the test case into 2 domains and run in parallel
- 2 files are written → `write()` function is executed by each process
- Task: ensure this function object just writes one file in parallel
- Static function `PStream::master()` → true if process runs on master node
- ⇒ wrap file streaming in if statement with `PStream::master()`

```
59 if (Pstream::master())
60 {
61     file_ << "# time" << token::TAB << "kineticEnergy" << endl;
62 }
```

- Repeat for the file streaming in `write()` function. Note that...
- `fvc::domainIntegrate` cannot be executed only on master node. Why?
- Compile and test in parallel → file written to `processor0` dir
- ⇒ change `fileName` in `OFstream` initialization

```
57 file_(obr_.time().rootPath()/obr_.time().globalCaseName()/name + ".dat")
```

root path

case directory name



- energy function object applicable to incompressible solvers only
- Task: extend applicability to compressible cases
- Compressible cases store varying density field — usually named rho
- ⇒ in write() function, set up logic based on rho field on database

```
100 scalar KE = 0.0;
101
102 if (obr_.foundObject<volScalarField>("rho"))
103 {
104     const volScalarField& rho = obr_.lookupObject<volScalarField>("rho");
105     KE = ...evaluation based on rho field... ;
106 }
107 else
108 {
109     KE = ...evaluation based on rho_ scalar... ;
110 }
111
112 if (Pstream::master())
113 {
114     file_ << obr_.time().timeName() << token::TAB << KE << endl;
115 }
```

- `rho` input is now redundant for compressible case
- ... assumed name of density field for compressible case
- Task: default `rho_ = 1`; add `rhoName_` entry, defaulting to "rho"
- Add private data `rhoName_` to `energy.H`
- Add "defaulting" initialization to `rhoName_` and `rho_` in `energy.C`

```
56 rhoName_(dict.lookupOrDefault<word>("rhoName", "rho")),
57 rho_(dict.lookupOrDefault<scalar>("rho", 1.0)),
```

- Modify `read()` function in `energy.C`

```
78 dict.readIfPresent("rhoName", rhoName_);
79 dict.readIfPresent("rho", rho_);
```

- Replace "rho" with `rhoName_` in object registry lookups
- Compile and test on a compressible flow case, e.g. `squareBend`
`$FOAM_TUTORIALS/compressible/rhoSimpleFoam/squareBend`

from Monitor.