High-level programming in OpenFOAM – and a first glance at C++

Solving PDEs with OpenFOAM

- The PDEs we wish to solve involve derivatives of tensor fields with respect to time and space
- The PDEs must be discretized in time and space before we solve them

- We will start by having a look at algebra of tensors in OpenFOAM at a single point
- We will then have a look at how to generate tensor fields from tensors
- Finally we will see how to discretize PDEs and how to set boundary conditions using high-level coding in OpenFOAM

- For further details, see the ProgrammersGuide

We will use 2.1.x, since we will use the `test` directory

Basic tensor classes in OpenFOAM

- Pre-defined classes for tensors of rank 0-3, but may be extended indefinitely

Rank	Common name	Basic name	Access function
0	Scalar	scalar	
1	Vector	vector	x(), y(), z()
2	Tensor	tensor	xx(), xy(), xz(), ...

Example:

A tensor $T = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$ is defined line-by-line:

```
tensor T( 11, 12, 13, 21, 22, 23, 31, 32, 33);

Info << "Txz = " << T.xz() << endl;
```

Outputs to the screen:

```
Txz = 13
```

Algebraic tensor operations in OpenFOAM

- Tensor operations operate on the entire tensor entity instead of a series of operations on its components
- The OpenFOAM syntax closely mimics the syntax used in written mathematics, using descriptive functions or symbolic operators

Examples:

Operation	Comment	Mathematical description	Description in OpenFOAM
Addition		$\mathbf{a} + \mathbf{b}$	a + b
Outer product	Rank $\mathbf{a}$, $\mathbf{b} \geq 1$	$\mathbf{ab}$	a * b
Inner product	Rank $\mathbf{a}$, $\mathbf{b} \geq 1$	$\mathbf{a} \cdot \mathbf{b}$	a & b
Cross product	Rank $\mathbf{a}$, $\mathbf{b} = 1$	$\mathbf{a} \times \mathbf{b}$	a ^ b
Operations exclusive to tensors of rank 2			
Transpose		$\mathbf{T}^T$	T.T()
Determinant		$\det\mathbf{T}$	det(T)
Operations exclusive to scalars			
Positive (boolean)		$s \geq 0$	pos(s)
Hyperbolic arc sine		asinh s	asinh(s)

Examples of the use of some tensor classes

- In `$FOAM_APP/test` we can find examples of the use of some classes.
- Tensor class examples:

```
run
cp -r $FOAM_APP/test .
cd test/tensor
wmake
Test-tensor >& log
```

- Have a look inside `Test-tensor.C` to see the high-level code.
- You see that `tensor.H` is included, which is located in `$FOAM_SRC/OpenFOAM/primitives/Tensor/tensor`. This defines how to compute eigenvalues. In `tensor.H`, `Tensor.H` is included (located in `$FOAM_SRC/OpenFOAM/primitives/Tensor`), which defines the access functions and includes `TensorI.H`, which defines the tensor operations.
- See also `vector`, `symmTensorField`, `sphericalTensorField` and many other examples.

Dimensional units in OpenFOAM

- OpenFOAM checks the dimensional consistency

Declaration of a tensor with dimensions:

```
dimensionedTensor sigma
    (
        "sigma",
        dimensionSet( 1, -1, -2, 0, 0, 0, 0),
        tensor( 1e6, 0, 0, 0, 1e6, 0, 0, 0, 1e6)
    );
```

The values of dimensionSet correspond to the powers of each SI unit:

No.	Property	Unit	Symbol
1	Mass	kilogram	kg
2	Length	metre	m
3	Time	second	s
4	Temperature	Kelvin	K
5	Quantity	moles	mol
6	Current	ampere	A
7	Luminous intensity	candela	cd

sigma then has the dimension $[kg/ms^2]$

Dimensional units in OpenFOAM

- Add the following to `Test-tensor.C`:
  Before `main()`:
  `#include "dimensionedTensor.H"`
  Before `return(0)`:

```
    dimensionedTensor sigma
    (
        "sigma",
        dimensionSet( 1, -1, -2, 0, 0, 0, 0),
        tensor( 1e6, 0, 0, 0, 1e6, 0, 0, 0, 1e6)
    );
    Info<< "Sigma: " << sigma << endl;
```

- Compile, run again, and you will get:

```
Sigma: sigma [1 -1 -2 0 0 0 0] (1e+06 0 0 0 1e+06 0 0 0 1e+06)
```

  You see that the object `sigma` that belongs to the `dimensionedTensor` class contains both the name, the dimensions and values.
- See `$FOAM_SRC/OpenFOAM/dimensionedTypes/dimensionedTensor`

Dimensional units in OpenFOAM

- Try some member functions of the `dimensionedTensor` class:

```
Info<< "Sigma name: " << sigma.name() << endl;
Info<< "Sigma dimensions: " << sigma.dimensions() << endl;
Info<< "Sigma value: " << sigma.value() << endl;
```

- You now also get:

```
Sigma name: sigma
Sigma dimensions: [1 -1 -2 0 0 0 0]
Sigma value: (1e+06 0 0 0 1e+06 0 0 0 1e+06)
```

- Extract one of the values:

```
Info<< "Sigma yy value: " << sigma.value().yy() << endl;
```

Note here that the `value()` member function first converts the expression to a `tensor`, which has a `yy()` member function. The `dimensionedTensor` class does not have a `yy()` member function, so it is not possible to do `sigma.yy()`.

Construction of a tensor field in OpenFOAM

- A tensor field is a list of tensors
- The use of typedef in OpenFOAM yields readable type definitions: scalarField, vectorField, tensorField, symmTensorField, ...
- Algebraic operations can be performed between different fields, and between a field and a single tensor, e.g. Field U, scalar 2.0: U = 2.0 * U;
- Add the following to `Test-tensor`:
  Before `main()`:
  `#include "tensorField.H"`
  Before `return(0)`:

```
tensorField tf1(2, tensor::one);
Info<< "tf1: " << tf1 << endl;
tf1[0] = tensor(1, 2, 3, 4, 5, 6, 7, 8, 9);
Info<< "tf1: " << tf1 << endl;
Info<< "2.0*tf1: " << 2.0*tf1 << endl;
```

Discretization of a tensor field in OpenFOAM

- FVM (Finite Volume Method)
- No limitations on the number of faces bounding each cell
- No restriction on the alignment of each face
- The mesh class polyMesh can be used to construct a polyhedral mesh using the minimum information required
- The fvMesh class extends the polyMesh class to include additional data needed for the FV discretization (see `test/mesh`)
- The geometricField class relates a tensor field to an fvMesh (can also be typedef volField, surfaceField, pointField)
- A geometricField inherits all the tensor algebra of its corresponding field, has dimension checking, and can be subjected to specific discretization procedures

Examine an fvMesh

- Let us examine an `fvMesh`:

```
run
rm -rf cavity
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity .
cd cavity
sed -i s/"20 20 1"/"2 2 1"/g constant/polyMesh/blockMeshDict
blockMesh
```

- Run `Test-mesh` (first compile it: `wmake $FOAM_RUN/test/mesh`)
- `C()` gives the center of all cells and boundary faces.
  `V()` gives the volume of all the cells.
  `Cf()` gives the center of all the faces.
- Try also adding in `Test-mesh.C`, before `return(0)`:

```
Info<< mesh.C().internalField()[1][1] << endl;
Info<< mesh.boundaryMesh()[0].name() << endl;
```

- See `$FOAM_SRC/finiteVolume/fvMesh`

Examine a volScalarField

- Read a `volScalarField` that corresponds to the `mesh`. Add in `Test-mesh.C`, before `return(0)`:

```
volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
Info<< p << endl;
Info<< p.boundaryField()[0] << endl;
```

Equation discretization in OpenFOAM

- Converts the PDEs into a set of linear algebraic equations, **Ax=b**, where **x** and **b** are volFields (geometricFields). **A** is an fvMatrix, which is created by a discretization of a geometricField and inherits the algebra of its corresponding field, and it supports many of the standard algebraic matrix operations
- The `fvm` (Finite Volume Matrix) and `fvc` (Finite Volume Calculus) classes contain static functions for the differential operators, and discretize any geometricField. `fvm` returns an `fvMatrix`, and `fvc` returns a `geometricField` (see `$FOAM_SRC/finiteVolume/finiteVolume/fvc` and `fvm`)

Examples:

Term description	Mathematical expression	fvm::/fvc:: functions
Laplacian	$\nabla \cdot \Gamma \nabla \phi$	laplacian(Gamma,phi)
Time derivative	$\partial \phi / \partial t$	ddt(phi)
	$\partial \rho \phi / \partial t$	ddt(rho, phi)
Convection	$\nabla \cdot (\psi)$	div(psi, scheme)
	$\nabla \cdot (\psi \phi)$	div(psi, phi, word)
		div(psi, phi)
Source	$\rho \phi$	Sp(rho, phi)
		SuSp(rho, phi)

ϕ: vol<type>Field, ρ: scalar, volScalarField, ψ: surfaceScalarField

Example

A call for solving the equation

$$\frac{\partial \rho \vec{U}}{\partial t} + \nabla \cdot \phi \vec{U} - \nabla \cdot \mu \nabla \vec{U} = -\nabla p$$

has the OpenFOAM representation

```
solve
(
        fvm::ddt(rho, U)
      + fvm::div(phi, U)
      - fvm::laplacian(mu, U)
        ==
      - fvc::grad(p)
)
```

Example: laplacianFoam, the source code

Solves $\partial T/\partial t - \nabla \cdot k\nabla T = 0$ (see `$FOAM_SOLVERS/basic/laplacianFoam`)

```
#include "fvCFD.H"  // Include the class declarations
int main(int argc, char *argv[])
{
#   include "setRootCase.H" // Set the correct path
#   include "createTime.H" // Create the time
#   include "createMesh.H" // Create the mesh
#   include "createFields.H" // Temperature field T and diffusivity DT
    while (runTime.loop()) // Time loop
    {
#   include "readSIMPLEControls.H" // Read solution controls
        for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
        {
            solve( fvm::ddt(T) - fvm::laplacian(DT, T) ); // Solve eq.
        }
#   include "write.H" // Write out results at specified time instances}
    }
    return 0; // End with 'ok' signal
}
```

Example: laplacianFoam, discretization and boundary conditions

See `$FOAM_TUTORIALS/basic/laplacianFoam/flange`

Discretization:

dictionary fvSchemes, read from file:

```
ddtSchemes
{
    default Euler;
}

laplacianSchemes
{
    default             none;
    laplacian(DT,T)     Gauss linear corrected;
}
```

Boundary conditions:

Part of class volScalarField object T, read from file:

```
boundaryField{
    patch1{ type zeroGradient;}
    patch2{ type fixedValue; value uniform 273;}}
```