How to implement a new boundary condition

How to implement a new boundary condition

- The implementations of the boundary conditions are located in `$FOAM_SRC/finiteVolume/fields/fvPatchFields/`
- To add a new boundary condition, start by finding one that does almost what you want. Copy that to your user directories under the same directory structure as the original installation.
- We will now check out the `parabolicVelocityFvPatchVectorField` boundary condition from the OpenFOAM-extend project at SourceForge, and compile and use it as a dynamic library.

Compile your boundary condition as a new dynamic library

- Copy the boundary condition to `$WM_PROJECT_USER_DIR` (from the OpenFOAM-extend project at SourceForge):

```
mkdir -p $WM_PROJECT_USER_DIR/src/finiteVolume/fields/fvPatchFields/derived
cd $WM_PROJECT_USER_DIR/src/finiteVolume/fields/fvPatchFields/derived
svn checkout svn://svn.code.sf.net/p/openfoam-extend/svn/trunk/\
Core/OpenFOAM-1.5-dev/src/finiteVolume/fields/fvPatchFields/\
derived/parabolicVelocity
cd $WM_PROJECT_USER_DIR/src/finiteVolume
```

- We need a Make/files file (c.f. `$FOAM_SRC/finiteVolume/Make`):

```
fvPatchFields = fields/fvPatchFields
derivedFvPatchFields = $(fvPatchFields)/derived
$(derivedFvPatchFields)/parabolicVelocity/parabolicVelocityFvPatchVectorField.C
LIB = $(FOAM_USER_LIBBIN)/libmyFiniteVolume
```

- We need a Make/options file:

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

- Compile the dynamic library:

```
wmake libso
```

Use your boundary condition from the dynamic library

- Set up a new case:

```
run
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily \
      pitzDailyParabolicInlet
cd pitzDailyParabolicInlet
blockMesh
```

- Modify the entry for the `inlet` boundary condition in `0/U` to:

```
type              parabolicVelocity;
n                 (1 0 0);
y                 (0 1 0);
maxValue          1;
value             uniform (0 0 0); // Dummy for paraFoam
```

The contents of this entry must be in accordance with the constructor in the `parabolicVelocityFvPatchVectorField` class. `n` is the direction of the flow, `y` is the coordinate direction of the profile, and maxvalue is the centerline velocity.

Use your boundary condition from the dynamic library

- The boundary condition will not be recognized by any of the original OpenFOAM solvers unless we tell OpenFOAM that the library exists.
  Add a line in the `system/controlDict` file:

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

  i.e. the library must be added for each case that will use it, but no re-compilation is needed for any solver. `libmyFiniteVolume.so` is found using the `LD_LIBRARY_PATH` environment variable, and if you followed the instructions on how to set up OpenFOAM and compile the boundary condition this should work automatically.
- You can now run the case using the original `simpleFoam` solver. Note that we never re-compiled the original `simpleFoam` solver, and if you do ``ldd `which simpleFoam` `` your new library will NOT show up since it is linked at run-time (using dlopen).

A look at the boundary condition

- The `parabolicVelocityFvPatchVectorField` boundary condition consists of two files:

```
parabolicVelocityFvPatchVectorField.C
parabolicVelocityFvPatchVectorField.H
```

- The `.H`-file is the header file, and it is included in the header of the `.C`-file.
- We can see (`.H`) that we create a sub class to the `fixedValueFvPatchVectorField`:

```
class parabolicVelocityFvPatchVectorField:
public fixedValueFvPatchVectorField
```

i.e. this is for Dirichlet (fixed) boundary conditions for vector fields.

- The class has the private data

```
//- Peak velocity magnitude
scalar maxValue_;
//- Flow direction
vector n_;
//- Direction of the y-coordinate
vector y_;
```

A look at the boundary condition

- The `TypeName("parabolicVelocity")`, used when specifying the boundary condition, is defined.
- There are some public constructors and member functions that are defined in detail in the `.C`-file.
- We used the third constructor when we tested the boundary condition, i.e. we read the member data from a dictionary.
- The actual implementation of the boundary condition can be found in the `updateCoeffs()` member function:

```
boundBox bb(patch().patch().localPoints(), true);
vector ctr = 0.5*(bb.max() + bb.min());
const vectorField& c = patch().Cf();
scalarField coord = 2*((c - ctr) & y_)/((bb.max() - bb.min()) & y_);
vectorField::operator=(n_*maxValue_*(1.0 - sqr(coord)));
```

A look at the boundary condition

- The member function `write` defines how to write out the boundary values in the time directory. The final line, `writeEntry("value", os);` writes out all the values, which is only needed for post-processing.
- Find out more about all the variables by including the following in the end of the `updateCoeffs` member function:

```
Info << "c" << c << endl;
Info << "ctr" << ctr << endl;
Info << "y_" << y_ << endl;
Info << "bb.max()" << bb.max() << endl;
Info << "bb.min()" << bb.min() << endl;
Info << "(c - ctr)" << c - ctr << endl;
Info << "((c - ctr) & y_)" << ((c - ctr) & y_) << endl;
Info << "((bb.max() - bb.min()) & y_)" <<
         ((bb.max() - bb.min()) & y_) << endl;
Info << "coord" << coord << endl;
```