Examples of how to use some utilities and functionObjects

(and some Gnuplot, Python, Matplotlib)

Some utilities and functionObjects

- We will now learn how to use a small number of useful utilities and functionObjects. Some of them are described in the UserGuide and ProgrammersGuide, some are described in the OpenFOAM Wiki (e.g. Turbomachinery Working Group) and some of them have been discussed in the Forum.

- It is HIGHLY recommended that you dig through ALL of the UserGuide and ProgrammersGuide (before complaining that there is not enough OpenFOAM documentation).

Postprocessing

- Some functionality that was earlier available as utilities for post processing have been reorganized using functionObjects (discussed more later).
- We will have a look at how to extract data for plotting and visualization in the coming slides.
- You can list the available options by typing

  ```
  postProcessing -list
  ```

  Use the `-help` flag for more information, as usual.
- You can figure out how to do other kinds of postProcessing by looking at info and examples in `$WM_PROJECT_DIR/etc/caseDicts/postProcessing/`

postProcess -func singleGraph

- `postProcess -func singleGraph` is used to produce graphs for publication.
- Copy the `singleGraph` dictionary from the `plateHole` tutorial:

```
cd $FOAM_RUN/icoFoam/cavity
cp $FOAM_TUTORIALS/stressAnalysis/solidDisplacementFoam/plateHole/system/singleGraph cavity/system
```

- Modify the `singleGraph` dictionary to make it match the `cavity` case:

```
sed -i s/"0 0.5 0.25"/"0.001 0.05 0.005"/g cavity/system/singleGraph
sed -i s/"0 2 0.25"/"0.099 0.05 0.005"/g cavity/system/singleGraph
sed -i s/"axis    y"/"axis    distance"/g cavity/system/singleGraph
sed -i s/"sigmaxx"/"p U"/g cavity/system/singleGraph
```

  Running `postProcess -func singleGraph -case cavity`, the variables `p` and `U` are extracted along a horizontal line at 100 points, and the results are written in `cavity/postProcessing/singleGraph`.

- Plot in `gnuplot` (first type `gnuplot` in a terminal window):
  `plot "cavity/postProcessing/singleGraph/0.5/line_p.xy"`
  `exit`
- Some info about plotting, in the two next slides...

Plotting with Gnuplot

- Read more about Gnuplot at `http://www.gnuplot.info/`
- Example of Gnuplot script (copy to `myPlot.gplt`, run with `gnuplot myPlot.gplt`, show plot with `display myPlot.png`):

```
set title "Velocity components"
set xlabel "Y-position"
set ylabel "Velocity"
set yrange [-0.3:0.3]
set xtics 0.02
set grid xtics noytics
set terminal png
set output "myPlot.png"
plot "cavity/postProcessing/singleGraph/0.5/line_U.xy"\
        using 1:2 every ::5 with lines title "U_X",\
     "cavity/postProcessing/singleGraph/0.5/line_U.xy"\
        using 1:3 every ::5 with lines title "U_Y"
```

- `using 1:2` means plot column 2 against 1
- `every ::5` means start at line 5
- Short format: `u 1:2 ev ::5 w l t "U_X"`

Plotting with Python and Matplotlib

- You can also use Python and Matplotlib to plot your results
- See http://openfoamwiki.net/index.php/Sig_Turbomachinery_/_Timisoara_Swirl_Generator#Post-processing_using_Python and http://www.scipy.org/Plotting_Tutorial
- Copy the text below to `plotPressure.py` (make sure the indentation is correct), type `python plotPressure.py`

```
#!/usr/bin/env python
description = """ Plot the pressure samples."""
import os, sys
import math
from pylab import *
from numpy import loadtxt
def addToPlots( timeName ):
    fileName = "cavity/postProcessing/singleGraph/" + timeName + "/line_p.xy"
    i=[]
    time=[]
    abc =loadtxt(fileName, skiprows=4)
    for z in abc:
        time.append(z[0])
        i.append(z[1])
    legend = "Pressure at " + timeName
    plot(time,i,label="Time " + timeName )
figure(1);
ylabel(" p/rho "); xlabel(" Distance (m) "); title(" Pressure along sample line ")
grid()
hold(True)
for dirStr in os.listdir("cavity/postProcessing/singleGraph/"):
    addToPlots( dirStr )
legend(loc="upper left")
savefig("myPlot.jpeg")
show() #Problems with ssh
```

Plotting with xmgrace

- You can plot the pressure with xmgrace:

```
xmgrace cavity/postProcessing/singleGraph/0.5/line_p.xy
```

- and the velocity components:

```
xmgrace -block \
cavity/postProcessing/singleGraph/0.5/line_U.xy -bxy 1:2 -bxy 1:3 -bxy 1:4
```

postProcess -func surfaces

- `postProcess -func surfaces` is used to extract surfaces in VTK format.
- Find a `surfaces` dictionary file example in
  `$WM_PROJECT_DIR/etc/caseDicts/postProcessing/visualization/surfaces`
  A modified version for the cavity case follows on the next slide, to be put in the system directory and executed by
  `postProcess -func surfaces -case cavity`
- The results are written in
  `cavity/postProcessing/surfaces`.
- Open the VTK files in `Paraview` / `paraFoam`

surfaces dictionary for postProcess -func surfaces

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Web:      www.OpenFOAM.org
     \\/     M anipulation  |
-------------------------------------------------------------------------------
Description
    Writes out surface files with interpolated field data in VTK format, e.g.
    cutting planes, iso-surfaces and patch boundary surfaces.

    This file includes a selection of example surfaces, each of which the user
    should configure and/or remove.

\*---------------------------------------------------------------------------*/

#includeEtc "caseDicts/postProcessing/visualization/surfaces.cfg"

fields       (p U);

surfaces
(
    zNormal
    {
        $cuttingPlane;
        pointAndNormalDict
        {
            basePoint    (0.05 0.05 0.005); // Overrides default basePoint (0 0 0)
            normalVector $z;      // $y: macro for (0 0 1)
        }
    }
    p0
    {
        $isosurface;
        isoField    p;
        isoValue    0;
    }
    movingWall
    {
        $patchSurface;
        patches     (movingWall);
    }
);

// ************************************************************************* //
```

postProcess -func 'div(U)' (and 'components(U)')

- Use `postProcess` to calculate new fields from existing ones.
- First add to `cavity/system/fvSchemes`, under `divSchemes`:
  `div(U) Gauss linear;`
  This is needed to specify how the `div` operator should operate.
- Execute by:
  `postProcess -func 'div(U)' -case cavity`
- Try also extracting the components of `U`:
  `postProcess -func 'components(U)' -case cavity`
- The new fields are written in the time directories.

The mapFields utility

- The `mapFields` utility maps the results of one case to another case. We already did the procedure while running the `icoFoam/Allrun` script, but let's do it again by hand:

```
cd $FOAM_RUN/icoFoam/cavity
#Assumes that you copied all icoFoam tutorials before:
#cp -r $FOAM_TUTORIALS/incompressible/icoFoam $FOAM_RUN
```

- Run the cavity case:

```
blockMesh -case cavity
icoFoam -case cavity >& log_cavity
```

- Prepare the `cavityClipped` case and map the `cavity/0.5` results to it:

```
blockMesh -case cavityClipped
cp -r cavityClipped/0 cavityClipped/0.5
mapFields cavity -case cavityClipped -sourceTime latestTime
```

- We first copied the `0` directory to `0.5`, since `mapFields` applies the mapping to the `startFrom/startTime` directory of the `cavityClipped` case, which is by default set to 0.5. Try setting `startTime 0;` to map to that time directory.
- The flag `-sourceTime latestTime` sais that the `cavity/0.5` results should be used.

The mapFields utility

- Type `mapFields -help` to get the optional flags
- The flag `-consistent` is used if the geometry and boundary conditions are identical in both cases. This is useful when modifying the mesh density of a case, while preserving the geometry and patch names.
- For non-consistent cases a `mapFieldsDict` dictionary must be edited, see `cavityClipped/system/mapFieldsDict`:

```
patchMap        ( lid movingWall );
cuttingPatches  ( fixedWalls );
```

  The first line sais that the name of the top patch has different names in the cases.
  The second line sais that the `fixedWalls` patch is cutting through the cavity case.

- The flags `-parallelSource` and `-parallelTarget` are used if any, or both, of the cases are decomposed for parallel simulations.

The setFields utility

- The `setFields` utility is used to set values to the fields in specific regions. You use this if you do the `interFoam/damBreak` tutorial in the UserGuide.
- Type `setFields -help` for optional flags
- A `setFieldsDict` dictionary is used. Find an example in the `damBreak` tutorial.
- We here copy the `setFieldsDict` from `damBreak`, and modify and run it for `cavity`:

```
cd $FOAM_RUN/icoFoam/cavity
cp $FOAM_TUTORIALS/multiphase/interFoam/ras/damBreak/damBreak/system/setFieldsDict cavity/system/
sed -i s/"alpha.water"/"p"/g cavity/system/setFieldsDict
sed -i s/"box (0 0 -1) (0.1461 0.292 1)"/"box (0 0 -1) (0.05 0.05 1)"/g cavity/system/setFieldsDict
setFields -case cavity
```

- Have a look at the `cavity/system/setFieldsDict`:
  - The `defaultFieldValues` sets the default values of the fields.
  - A `boxToCell` bounding box is used to define a set of cells where the `fieldValues` should be different than the `defaultFieldValues`.
  - Use a `dummy` instead of `boxToCell` to see the `topoSetSource` alternatives.

The funkySetFields, groovyBC and swak4Foam utilities (related to setFields)

These are really useful community contributions!

- `funkySetFields` is a development of the `setFields` utility, and it includes the option of specifying mathematical expressions etc.:
  `http://openfoamwiki.net/index.php/Contrib_funkySetFields`

- The `groovyBC` utility is similar, but for boundaries:
  `http://openfoamwiki.net/index.php/Contrib_groovyBC`
  It should be noted that from 2.0.x, there is a new way of setting boundary conditions similar to `groovyBC`, but with C++ syntax (`codedFixedValue` - google it!).

- The above have now been merged into `swak4Foam` (Swiss Army Knife For Foam):

  `http://openfoamwiki.net/index.php/Contrib/swak4Foam`

  See also the OpenFOAM Workshop training material:

  `www.openfoamworkshop.org`

The foamToVTK, checkMesh, and flattenMesh utilities

- The foamToVTK utility can be used in many different ways. Example:
- The two empty sides of a 2D mesh must have the same mesh distribution. Add `0.0001` to the z-position of one of the `constant/polyMesh/points` of the `cavity` case.
- The `checkMesh` utility can be used to verify this. If not, it will complain:

```
***Number of edges not aligned with or perpendicular to non-empty directions: ????
Writing ???? points on non-aligned edges to set nonAlignedEdges
```

- The point labels are written to `constant/polyMesh/sets/nonAlignedEdges`
- Take the opportunity to visualize the point set in `paraFoam`: First open the cavity case in `paraFoam`, then use File/Open `<case>.OpenFOAM` to read in the same case again. This time mark `Include Sets`, mark *only* Mesh Parts/NonAlignedEdges, and visualize using box glyphs.
- Another way to view the problematic points in *paraview* (not `paraFoam`):

```
foamToVTK -case cavity -pointSet nonAlignedEdges
```

  The result appears in the VTK directory.

- The `flattenMesh` utility can sometimes fix the problem, like in this case.

The transformPoints utility

- Moves, rotates and scales the mesh.
- Usage (`transformPoints -help`, version dependent):

```
transformPoints [-translate vector] [-yawPitchRoll (yaw pitch roll)]
    [-rotateFields] [-parallel] [-rotate (vector vector)]
    [-rollPitchYaw (roll pitch yaw)] [-scale vector] [-case dir]
    [-help] [-doc] [-srcDoc]
```

- Example:

```
cd $FOAM_RUN/icoFoam/cavity; blockMesh -case cavity
cp -r cavity cavityMoved
transformPoints -case cavityMoved -translate "(0.1 0 0)"
```

- Have a look in paraFoam:

```
touch cavityMoved/cavityMoved.OpenFOAM
paraFoam -case cavity
```

Click Apply and then use File/Open to open the `cavityMoved.OpenFOAM` file at the same time.

The mergeMeshes utility

- Takes the meshes from two different cases and merges them into the master case.
- `mergeMeshes` reads the `system/controlDict` of both cases and uses the `startTime`, so be careful if you have a moving mesh for example. The first case that you specify will be the master, and a new time (`startTime`+`deltaT`) will be written in which a new polymesh is located. Move it to the correct position (`constant/polyMesh`), and you have a case with the merged mesh.
- Example (start from clean cases):

```
run
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity/cavity cavityMerged
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity/cavity cavityTransformed
blockMesh -case cavityMerged
blockMesh -case cavityTransformed
transformPoints -case cavityTransformed -translate "(0.1 0 0)"
mergeMeshes cavityMerged cavityTransformed
mv cavityMerged/0.005/polyMesh/* cavityMerged/constant/polyMesh
```

- Note that the two meshes will keep all their original boundary conditions, so they are not automatically coupled. Try `icoFoam`! To couple the meshes, use `stitchMesh`...

The stitchMesh utility

- Couples two uncoupled mesh regions, belonging to the same case.
- You should have a patch in one region of the mesh (`masterPatch`) that fits with a corresponding patch in the other region of the mesh (`slavePatch`). If you have that, then the command is:
  `stitchMesh masterPatch slavePatch`
- After `stitchMesh`, `masterPatch` and `slavePatch` are still present in the new `polymesh/boundary`, but they are empty so just delete them. The same thing can be done as well for the boundary conditions in the `0` folder.
- We have to re-organize the patches for this to work with our `cavityMerged` case, so we will do it for another case.

First let's run and have a look at the original `interFoam/ras/damBreak/damBreak` tutorial
(not in slides)

Example: mergeMeshes and stitchMesh

Create two fresh `interFoam/ras/damBreak` cases:

```
run
cp -r $FOAM_TUTORIALS/multiphase/interFoam/ras/damBreak/damBreak damBreakLeft
cp -r $FOAM_TUTORIALS/multiphase/interFoam/ras/damBreak/damBreak damBreakRight
```

Change the right wall name of `damBreakLeft` to `rightWallLeft`, and the left wall name of `damBreakRight` to `rightWallRight`, since they are to be stitched (make sure to do this only once!):

```
sed -i s/rightWall/rightWallLeft/g damBreakLeft/system/blockMeshDict
sed -i s/leftWall/leftWallRight/g damBreakRight/system/blockMeshDict
```

Modify the number of cells in `damBreakRight` and create the meshes of both parts.:

```
sed -i s/" 42 "/" 43 "/g damBreakRight/system/blockMeshDict
blockMesh -case damBreakLeft
blockMesh -case damBreakRight
```

Example: mergeMeshes and stitchMesh

Move the `damBreakRight` case so that its `leftWallRight` coincides with the `rightWallLeft` patch of `damBreakLeft`, and merge the meshes into `damBreakLeft`:

```
transformPoints -translate "(0.584 0 0)" -case damBreakRight
mergeMeshes damBreakLeft damBreakRight
rm -r damBreakRight
rm -r damBreakLeft/constant/polyMesh
mv damBreakLeft/0.001/polyMesh damBreakLeft/constant
rmdir damBreakLeft/0.001
```

Change the patch names in the `0`-directory, using regex(7) POSIX expressions (make sure to do this only once!):

```
sed -i s/"leftWall"/'"leftWall.*"'/g damBreakLeft/0/*
sed -i s/"rightWall"/'"rightWall.*"'/g damBreakLeft/0/*
```

Run the case:

```
setFields -case damBreakLeft
interFoam -case damBreakLeft >& log
paraFoam -case damBreakLeft
```

Clean up:

```
rm -r damBreakLeft/{0.*,[1-9]*}
```

Example: mergeMeshes and stitchMesh

Stitch the `damBreakLeft` case:

- Check that you have two regions:

```
checkMesh -case damBreakLeft #=> *Number of regions: 2
```

- Stitch the two regions into a single region:

```
stitchMesh -case damBreakLeft rightWallLeft leftWallRight
rm -r damBreakLeft/constant/polyMesh
mv damBreakLeft/0.001/polyMesh damBreakLeft/constant
rm -r damBreakLeft/0.001 #Also the variables have been saved, but we
                         #keep the original in the 0 directory!
```

- Check that you have one region:

```
checkMesh -case damBreakLeft #=> Number of regions: 1 (OK).
```

- Run and visualize (use Surface With Edges representation to see the stitching):

```
setFields -case damBreakLeft
interFoam -case damBreakLeft >& log
paraFoam -case damBreakLeft
```

The decomposePar utility

- `decomposePar` makes a domain decomposition for parallel computations. This is described in the UserGuide.
- Type `decomposePar -help` to see optional flags
- A `decomposeParDict` specifies how the mesh should be decomposed. An example can be found in the `interFoam/damBreak` tutorial: `system/decomposeParDict`. `numberOfSubdomains` specifies the number of subdomains the grid should be decomposed into. Make sure that you specify the same number of subdomains in the specific decomposition method you will use, otherwise your simulation might not run optimal.
- Try running in parallel:

```
rm -r damBreakLeft/{0.*,[1-9]*}
decomposePar -case damBreakLeft
mpirun -np 4 interFoam -case damBreakLeft -parallel >& log&
top
```

- Use flag `-force` if you have already decomposed, but want to do it again.
- Try also changing to `scotch` (does not require a `scotchCoeffs`)

The reconstructPar utility

- `reconstructPar` is the reverse of `decomposePar`, reassembling the mesh and the results.
- Type `reconstructPar -help` to see optional flags
- This is usually done for post-processing, although it is also possible to post-process each domain separately by treating an individual processor directory as a separate case when starting `paraFoam, or using the paraFoam flag \verb`-builtin+.
- Try reconstructing our case:

  ```
  reconstructPar -case damBreakLeft
  ```

  It will do the time directories that are available until now.

Modifying dictionaries with changeDictionary

In `system/changeDictionaryDict` (test on clean `cavity` case):

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      changeDictionaryDict;
}
boundary
{
    frontAndBack
    {
        type            symmetryPlane;
    }
}
U
{
    internalField   uniform (0.01 0 0);
    boundaryField
    {
        frontAndBack
        {
            type            symmetryPlane;
        }
        ".*Wall.*" // ".*" is for RegExp
        {
            type            fixedValue;
            value           uniform (0.01 0 0);
        }
    }
}
```

functionObjects

- functionObjects are general libraries that can be attached run-time to any solver, without having to re-compile the solver.
- An example can be found in the `incompressible/pisoFoam/les/pitzDaily` tutorial.
- A functionObject is added to a solver by adding a `functions` entry in `system/controlDict`
- You can find `functionObjects` in the source code, in the OpenFOAM Wiki (`www.openfoamwiki.net`), and in the OpenFOAM-extend project (`www.sourceforge.net`).
- The implementations can be found in:
  `$FOAM_SRC/functionObjects`
  Have a look in the `*.H` files for descriptions of how to use them (see e.g. `forces/forces/forces.H`). This information can also be found in Doxygen. In some cases there are also `controlDict` examples in those directories.

The fieldMinMax functionObject

- Add to `damBreakLeft/system/controlDict`:

```
functions
{
    minMaxU
    {
        type        fieldMinMax;
        libs ("libfieldFunctionObjects.so");
        enabled true;
        log false;
        write true;
        fields
        (
            U
        );
        mode        magnitude;
        writeControl   timeStep;
        writeInterval  1;
    }
}
```

- Run the case:

```
interFoam -case damBreakLeft >& log&
```

- Output in (time directory according to when it was initialized):

```
damBreakLeft/postProcessing/minMaxU/0/fieldMinMax.dat
```

Plot the output of fieldMinMax

- The output of `fieldMinMax` is a bit complex:

```
# Field minima and maxima
# Time  field   min     position(min)   max     position(max)
0.00119048      U       0       (0 0.00299993 0.0073)   0.0553571       (0.146 0.296857 0.0073)
0.00258503      U       0       (0 0.00299993 0.0073)   0.124416        (0.171391 0.00299994 0.0073)
0.00422003      U       0       (0 0.00299993 0.0073)   0.238544        (0.171391 0.00299994 0.0073)
```

- Use `sed` and `gnuplot` to plot, removing headerline and unwanted characters (`U`, `(`, and `)`):

```
plot '<sed "s/U//g;s/(//g;s/)//g" damBreakLeft/postProcessing/minMaxU/0/fieldMinMax.dat'\
 using 1:7 every ::2 with lines title "X-position of maximum velocity magnitude"
```

The probes functionObject

- The probes functionObject probes the development of the results during a simulation, writing to a file in the directory `postProcessing/probes`.
- Be inspired by `incompressible/pisoFoam/les/pitzDaily` and add to the `functions` dictionary in `controlDict`:

```
probes
{
    type            probes;
    functionObjectLibs ("libsampling.so");
    enabled         true;
    writeControl    timeStep;
    writeInterval   1;
    fields
    (
        U
    );
    probeLocations
    (
        ( 0.1778 0.0253 0 )
    );
}
```

- Plot with gnuplot as for `fieldMinMax`
- Note that the values are the cell center values, i.e. not interpolated!

The fieldAverage functionObject

- The `fieldAverage` functionObject calculates the time-average of specified fields and writes the results in the time directories.
- Be inspired by `incompressible/pisoFoam/les/pitzDaily` and add to the `functions` dictionary in `controlDict`:

```
fieldAverage1
{
    type            fieldAverage;
    libs ("libfieldFunctionObjects.so");
    enabled         true;
    writeControl    outputTime;
    fields
    (
        U
        {
            mean        on;
            prime2Mean  on; //RMS
            base        time;
        }
    );
}
```

- There are now also files `fieldAverage1Properties` in `[0-9]*/uniform`.

The surfaces functionObject

The surfaces functionObject writes out surface interpolated results to disk.
If the `surfaceFormat` is VTK, those can be viewed in paraview.

First example:

```
surfaceSampling
{
    type surfaces;
    libs ("libsampling.so");
    enabled         true;
    writeControl   outputTime;
    interpolationScheme cellPoint;
    surfaceFormat vtk;
    fields ( U );
    surfaces
    (
        nearWall
        {
            type            patchInternalField;
            patches         ( leftWall );
            distance        1E-6;
            interpolate     true;
            triangulate     false;
        }
    );
}
```

Two more examples:

```
atmosphere
{
    type            patch;
    patches       ( atmosphere );
    triangulate     false;
}
plane
{
    type            plane;
    normalVector (0 0 1);
    basePoint (0 0 0.005);
}
```

The VTK files end up in `postProcessing/surfaceSampling` for each `outputTime`.

The forces functionObject

- The viscous and pressure forces and moments (about a center of rotation) is reported by the `forces` functionObject (try with `damBreakLeft`):

```
forces
{
    type forces;
    libs ("libforces.so");
    patches ( lowerWall );
    rhoName rho;
    pName p;
    UName U;
    CofR (0 0 0);
    rhoInf 1000;
    name forces;
    uitype forces;
    writeControl timeStep;
    writeInterval 1;
    format ascii;
}
```

The forceCoeffs functionObject

- The lift and drag coefficients are reported by the `forceCoeffs` functionObject (try with `damBreakLeft`), see `sonicFoam/ras`:

```
forceCoeffs
{
    type            forceCoeffs;
    libs ( "libforces.so" );
    writeControl   timeStep;
    writeInterval  1;
    patches ( lowerWall );
    pName       p;
    UName       U;
    log         true;
    rhoInf      1;
    CofR        ( 0 0 0 );
    liftDir     ( -0.239733 0.970839 0 );
    dragDir     ( 0.970839 0.239733 0 );
    pitchAxis   ( 0 0 1 );
    magUInf     618.022;
    lRef        1;
    Aref        1;
}
```

More functionObjects

- http://openfoamwiki.net/index.php/Contrib_simpleFunctionObjects
- http://openfoamwiki.net/index.php/Sig_Turbomachienry_/_ERCOFTAC_centrifugal_pump_with_a_vaned_diffuser#Optional_tools
- http://openfoamwiki.net/index.php/Contrib/swak4Foam