pyFoam

Productive use of OpenFOAM from the command line

Bernhard F.W. Gschaider

Jeju, Korea
11. June 2013

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

Outline

Intended audience and aim

- Intended audience for this presentation:
  - people who already worked a bit with OpenFOAM

    worked a bit been through the tutorials and set up a case on their own
  - not yet experts with PyFoam
    - but even people who already worked with PyFoam will learn something new
- Aim of the presentation
  - Enable user to efficiently use PyFoam
  - No programming
- The presentation is designed so that all steps can be reproduced using the information on the slides

Motto

GUIs are OK.
But talking is a faster way of communication than pointing

- GUIs have their uses
  - Are more intuitive on first use
- Some things are faster on the command line
  - Once you climbed the "learning curve"

Outline

What is PyFoam

- PyFoam is a library for
  - Manipulating OpenFOAM-cases
  - Controlling OpenFOAM-runs
- It is written in Python
- Based upon that library there is a number of utilities
  - For case manipulation
  - Running simulations
  - Looking at the results
- All utilities start with pyFoam (so TAB-completion gives you an overview)
  - Each utility has an online help that is shown when using the --help-option
  - Additional information can be found
    - on openfoamwiki.net

Previous presentations

- PyFoam - Happy foaming with Python: Held at the OpenFOAM-Workshop in Montreal (2009) Introduction of PyFoam - mostly about the utilities
- Automatization with pyFoam - How Python helps us to avoid contact with OpenFOAM: Held at the Workshop in Gothenburg (2010)
  Mainly about writing scripts with PyFoam
- pyFoam - The dark, unknown corners: Held at the 2012 PFAU (Austrian User Meeting).
  About the tools for quantitative analysis
- Automatic testing of solvers using PyFoam: Held at the Workshop in Darmstadt (2012)
  About a very specific use of PyFoam that involves programming

Outline

Who is Ignaz

- **Ignaz Gartengschirrl** is a CFD-engineer working at Dambreakers Inc.
  - He is a specialist on the `damBreak`-case
- We met him in previous presentations
  - About PyFoam but also swak4Foam
- His company is currently extending their operations from two-phase dambreaking to multi-phase dambreaking

What Ignaz does in this presentation

We'll accompany Ignaz on two typical work days

1. On the first day Ignaz is looking into a solver that is new to him: `multiphaseInterFoam`
   - he experiments with a tutorial case that is quite similar to his previous work
     - write out some additional information using standard OpenFOAM-functionObjects
   - while he does so he will use a wide range of utilities
2. On the second day Ignaz will lead a project to adapt the `damBreak`-case for a very different application
   - he will collaborate with two colleagues. Each of them improving the case in his field of expertise
   - the will use tools in PyFoam (and others) to make the collaboration less painful

Outline

Command line examples

- In the following presentation we will enter things on the command line. Short examples will be a single line (without output)

```
ls $HOME
```

- Long examples will be a white box
  - Input will be prefixed with a > and blue
  - Long lines will be broken up
    - A pair of <brk> and <cont> indicates that this is still the same line in the input/output

```
> this is an example for a very long command line <brk>
    <cont>that does not fit onto one line of the <brk>
    <cont>slide but we have to write it anyway
first line of output (short)
Second line of output which is too long for this <brk>
    <cont>slide but we got to read it in all its <brk>
    <cont>glory
```

Data files in PyFoamTraining_AdditionalFiles

- A tar file with the required files in /home/openfoam/training_materials/Track3-2
  - unpack to home-directory

```
> cd $HOME; tar xvzf /home/openfoam/<brk>
<cont>training_materials/Track3-2/<brk>
<cont>PyFoamTrainingMaterials.tgz
```

- Contents of the directory PyFoamTrainingMaterials

  dotAndMercurialExtensions.tgz a tar-file with some dot-files and extensions for mercurial

  IHavePreparedSomething Collection of files for quick copying to short-cut lengthy file operations

Adding stuff to the environment

- This operation copies some files to your local environment.
  - Should not overwrite anything that came with the stick
    - You don't have to do this. **Most** (but not all) things will work anyway
- Added files:

.zshrc	Configuration file from http://grml.org/ that makes `zsh` really great
.zshrc.local	Additional setting to display currently used OpenFOAM-version
mercurialExtensions	directory with additional extension for `mercurial`
.hgrc	Settings-file for mercurial that uses the above extensions

- Install the files

```
1 > cd $HOME; tar xvzf ~/PyFoamTrainingMaterials/<brk>
    <cont>dotAndMercurualExtensions.tgz
```

Getting onto the same page

- During the remaining presentation we assume that
  - the zsh is used (optional)
  - we use OpenFOAM 2.2 (required)
- Switch to zsh

```
zsh
```

- You should see a more colorful prompt with (OF:-) on the left
- Switch on OpenFOAM 2.2

```
. /opt/openfoam220/etc/bashrc
```

- Now the prompt should show (OF:2.2.0-Opt)
- Create a working directory and go there

```
mkdir PyFoamTraining; cd PyFoamTraining
```

Make sure pyFoam is working

- There is a utility that helps make sure that PyFoam is working
  - and gives valuable information for support

```
> pyFoamVersion.py
Machine info: Darwin | bgs-cool-greybook.local | 12.3.0 | Darwin
Kernel Version 12.3.0: Sun Jan  6 22:37:10 PST 2013; root:
xnu-2050.22.13~1/RELEASE_X86_64 | x86_64 | i386

Python version: 2.7.3 (default, Nov 15 2012, 19:15:30)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apple/clang-421.11.66))
]

Python executable: /opt/local/Library/Frameworks/Python.framework/
Versions/2.7/Resources/Python.app/Contents/MacOS/Python

PYTHONPATH: /opt/local/lib/vtk-5.2/:/Users/bgschaid/private_python:

Location of this utility: /Users/bgschaid/Development/OpenFOAM/
Python/PyFoam/bin/pyFoamVersion.py

OpenFOAM (2, 2, 'x') of the installed versions ['2.1.x', '1.7.x',
'1.6.x', '1.6-ext', '2.2.x', '2.0.x']

pyFoam-Version: 0.6.1-development

Path where PyFoam was found (PyFoam.__path__) is ['/Users/bgschaid/
private_python/PyFoam']

Configuration search path: [('file', '/etc/pyFoam/pyfoamrc'), ('
directory', '/etc/pyFoam/pyfoamrc.d'), ('file', '/Users/
bgschaid/.pyFoam/pyfoamrc'), ('directory', '/Users/bgschaid
/.pyFoam/pyfoamrc.d')]
Configuration files (used): ['/Users/bgschaid/.pyFoam/pyfoamrc', '/
Users/bgschaid/.pyFoam/pyfoamrc.d/testit.cfg']
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Information by pyFoamVersion.py

- Machine
- Used python
- PYTHONPATH (where additional libraries are searched)
- Information about the used PyFoam
  - Where configuration files are sought
- Installed libraries relevant for PyFoam
  - With version if possible

Outline

Outline

The case we use

- Tutorial-case for the `multiphaseInterFoam`-solver `damBreak4phase`:
  - Extension of the usual `damBreak` with 4 different fluids:

air	light blue in the following pictures
water	dark blue
oil	dark grey
mercury	light grey

Initial conditions

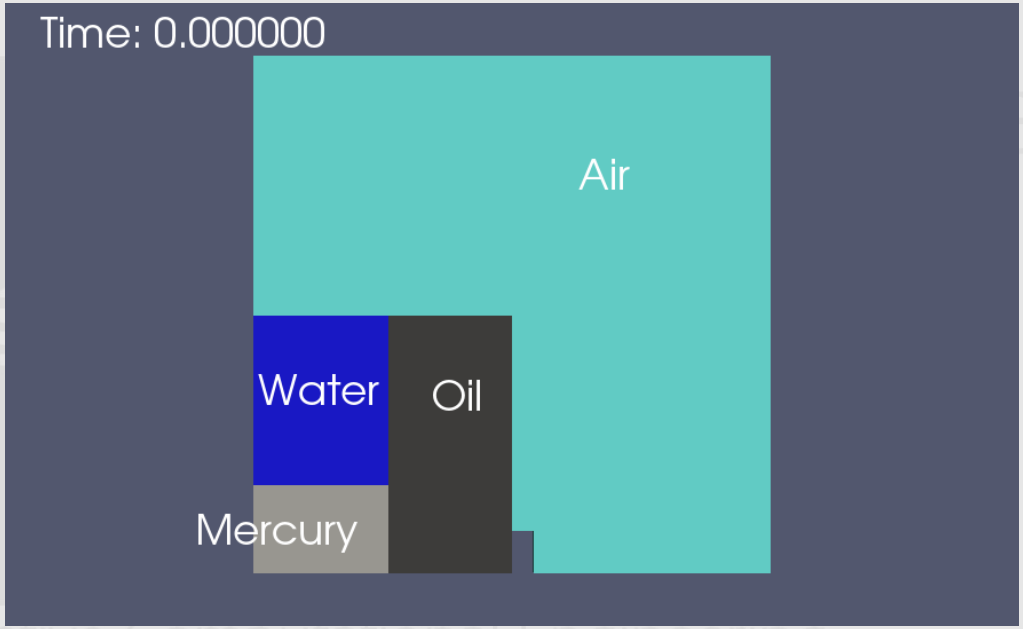


Figure : Fluids in the beginning

Getting the basic case

- Define a variable for convenient access

```
> export BCASE=$FOAM_TUTORIALS/multiphase/<brk>
  <cont>multiphaseInterFoam/laminar/<brk>
  <cont>damBreak4phase
```

- Usually we'd copy the case like this (**don't do it**):

```
cp -r $BCASE .
```

- But we use our first PyFoam-command:

```
pyFoamCloneCase.py $BCASE damBreakStart
cd damBreakStart
```

Files copied and created by `pyFoamCloneCase.py`

- The utility only copies files that are necessary to run the case
  - `system` and `constant` directory
  - `0` and/or `0.org`
  - `Allrun` and `Allclean`
- Also certain PyFoam-specific files (`customRegexp` etc)
- Additional files/directories can be specified
- Two files are created

  PyFoamHistory log-file with all things done to the case by PyFoam

  <casename>.foam a stub-file that is used by the built-in Paraview-reader

Preparing the case

We do this without PyFoam

- run `blockMesh`

```
blockMesh
```

- copy over the saved initial conditions

```
rm -rf 0 ; cp -r 0.org 0
```

- set initial fields

```
setFields
```

Running the case for the first time

- The `Runner`-utilities in PyFoam take options in this order:
  1. Utility name
  2. Specific options for the utility (optional)
  3. OpenFOAM solver/utility
  4. Options for that solver/utility (optional)
- We run the solver for the first time

```
pyFoamPlotRunner.py multiphaseInterFoam
```

- We see the usual output
- Two windows with plots

Residual

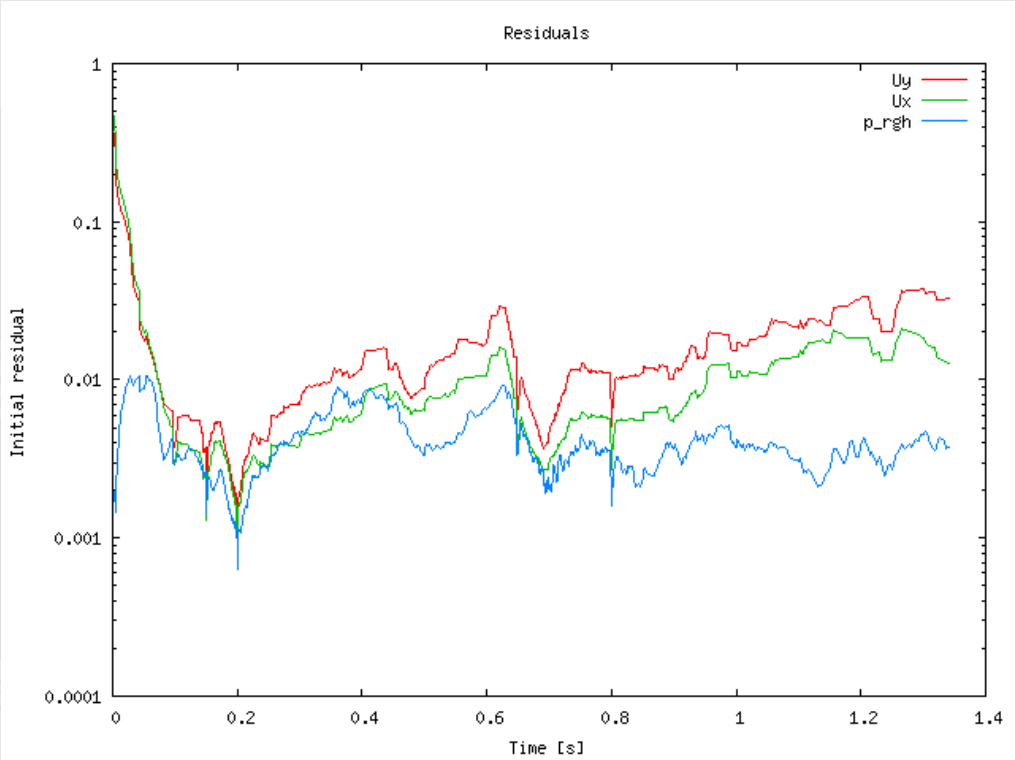


Figure : Residual plot with logarithmic scale

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Continuity

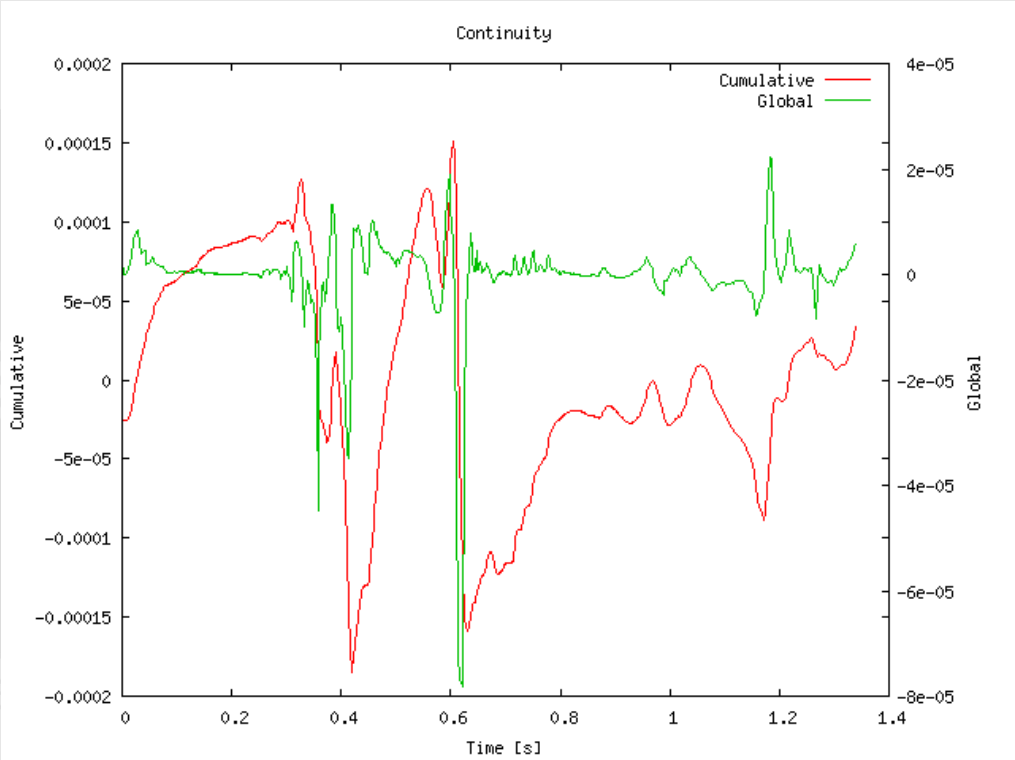


Figure : Continuity plot with two separate scales

Files and directories PyFoam added

- When we look at the directory we see some additional stuff
- most of it suspiciously starting with `PyFoam`

PyFoamRunner.multiphaseInterFoam.logfile File with all the output that went to the screen (usually named after the solver)

PyFoamState.* Files with the current state as plain text

  - names explain what is in there
  - Used by `pyFoamListCases.py` (later)

PyFoamServer.info Information about where a server process can be accessed (more about that later)

Gnuplotting.analyzed Directory that has the data `PyFoam` analyzed until now in pickled format

Excurse: Pickled

- **Pickled** is a Python-specific file format
  - Stores Python-structures
  - Can be easily loaded back into memory
- Advantages
  - Easily used in any Python-program
  - Sufficiently fast
- Disadvantage
  - only readable by Python-programs
- It is used in PyFoam
  - For storing data
  - To allow PyFoam-utilities to communicate with each other
    - Through UNIX-pipes or over the network

Contents of the `*.analyzed`-directory

pickledPlots Data to reproduce the plots in pickled format (written at intervals)

pickledStartData Data about the run at the beginning

pickledUnfinishedData The above data plus the latest result (written at intervals)

pickledData The collected data in the end

Reading this data enables other scripts to easily use information about the run

- the directory may hold additional plain-text versions of the analyzed data if specified

Printing information about the run

```
> pyFoamEchoPickledApplicationData.py --pickled-file=Gnuplotting.analyzed/<brk>
    <cont>pickledUnfinishedData --print-data
{'OK': True,
 'analyzed': {'Continuity': {'Cumulative': 3.34653e-05, 'Global': 5.7314e-06},
              'Courant': {'max': 0.47448, 'mean': 0.102383},
              'DeltaT': {'deltaT': 0.00321429},
              'Execution': {'clock': 0.0, 'cpu': 0.060000000000002274},
              'Iterations': {'Ux': 7.0, 'Uy': 8.0, 'p_rgh': 6.0},
              'Linear': {'Ux': 0.0128614,
                         'Ux_final': 2.51218e-09,
                         'Ux_iterations': 7.0,
                         'Uy': 0.0328763,
                         'Uy_final': 3.12009e-09,
                         'Uy_iterations': 8.0,
                         'p_rgh': 0.00373021,
                         'p_rgh_final': 3.58528e-08,
                         'p_rgh_iterations': 6.0}},
 'casefullname': '/Volumes/Foam/Cases/Scratch4KoreaPyFoam/damBreakStart',
 'casename': 'damBreakStart',
 'commandLine': 'multiphaseInterFoam',
 'cpuSystemTime': 0.0,
 'cpuTime': 0.0,
 'cpuUserTime': 0.0,
 'endtime': 'Mon May 20 17:35:18 2013',
 'fatalError': False,
 'fatalFPE': False,
 'fatalStackdump': False,
 'hostname': 'bgs-cool-greybook.local',
 'interrupted': False,
 'lasttimesteptime': 'Mon May 20 17:35:18 2013',
 'lines': 36920,
 'logfile': './PyFoamRunner.multiphaseInterFoam.logfile',
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Reproducing the plots from the stored data

```
> pyFoamRedoPlot.py --pickle-file Gnuplotting.<brk>
    <cont>analyzed/pickledPlots
 PyFoam WARNING on line 152 of file /Users/bgschaid/<brk>
     <cont>private_python/PyFoam/Applications/<brk>
     <cont>RedoPlot.py : Only the first parameter is <brk>
     <cont>used
Found 3 plots and 7 data sets
Adding line 1
Adding line 3
Adding line 2
Adding line 5
Adding line 4
Adding line 7
Adding line 6
Plotting 1 : linear
Plotting 3 : bounding No data - skipping
Plotting 2 : continuity
```

And we get two pictures in the directory

Redone residuals `linear.png`

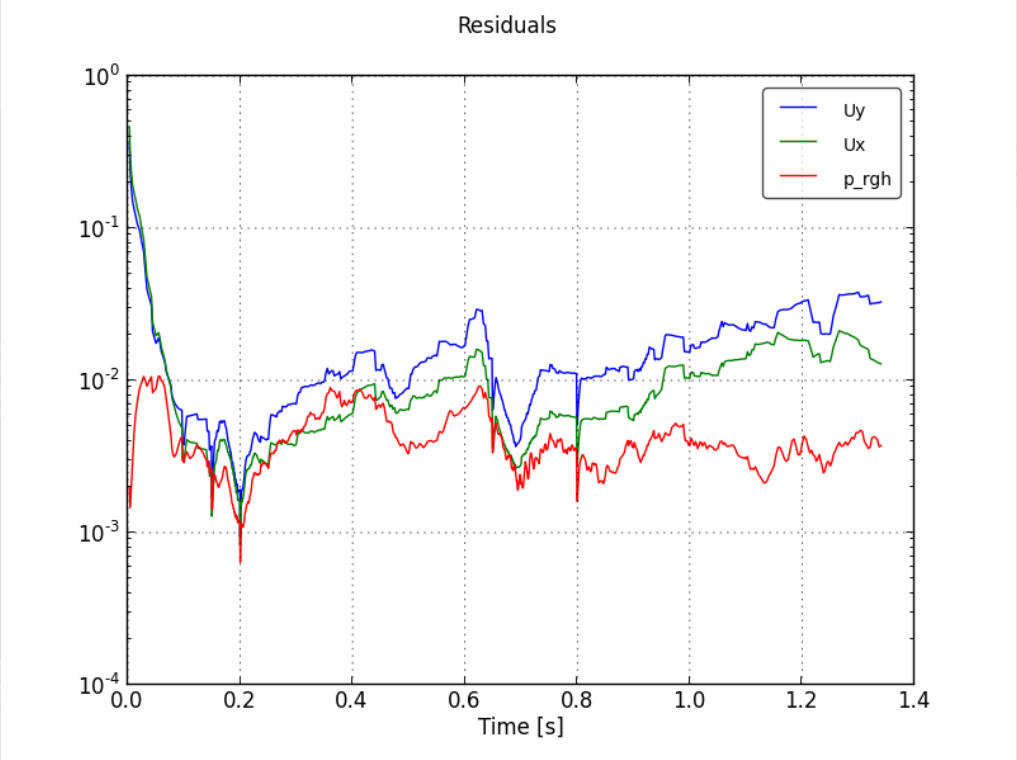


Figure : Residuals plotted with `matplotlib`

Writing the data with pyFoamRedoPlot.py

```
> pyFoamRedoPlot.py --csv-files --pickle-file Gnuplotting.<brk>
    <cont>analyzed/pickledPlots
...
> less continuity.csv
time,Cumulative,Global
2.58502999999999880e-03,-2.59226999999999831e<brk>
    <cont>-05,1.06142000000000038e-06
4.22002999999999615e-03,-2.57956000000000083e<brk>
    <cont>-05,1.21327000000000117e-07
6.12752999999999956e-03,-2.57906000000000003e<brk>
    <cont>-05,1.54292999999999841e-08
8.32114999999999380e-03,-2.54419000000000115e<brk>
    <cont>-05,3.65102999999999838e-07
1.09260999999999935e-02,-2.43567000000000009e<brk>
    <cont>-05,1.08202000000000103e-06
...
```

- CSV-files can be easily imported into other programs (spreadsheets for instance)

Less printing more plotting

- Lets start the run again

```
> pyFoamPlotRunner.py --clear-case --hardcopy <brk>
    <cont>--progress --with-all <brk>
    <cont>multiphaseInterFoam
2   t =     0.469879
```

- Additional options here:
  - **–clear-case** Remove all timesteps except the first before running
  - **–progress** Only print the time to screen (output will go to the logfile anyway)
  - **–with-all** Plot additional information like the timestep-size
  - **–hardcopy** in the save bitmaps of the plots

Dealing with the aftermath

- `pyFoamPlotRunner.py` may leaves the plot windows open
  - Which is good, because you might want to look at them after run
  - But it may be tedious to close them all
- This closes all windows
  - But be careful: it kills **all** Gnuplot windows

```
killall gnuplot_x11
```

- Afterwards we can look at the saved hardcopies

```
display *.png
```

Getting help

Utilities like `pyFoamPlotRunner.py` have loads of options:

```
> pyFoamPlotRunner.py --help
Usage
=====
  pyFoamPlotRunner.py [options] <foamApplication> [foamOptions]

Runs an OpenFoam solver needs the usual 3 arguments (<solver> <<brk>
    <cont>directory>
<case>) and passes them on (plus additional arguments). Output is <brk>
    <cont>sent to
stdout and a logfile inside the case directory (PyFoamSolver.logfile<brk>
    <cont>)
Information about the residuals is output as graphs  If the <brk>
    <cont>directory contains
a file customRegexp this is automatically read and the regular <brk>
    <cont>expressions in
it are displayed

Options
=======
--version                 show program's version number and exit
--help, -h                show this help message and exit
--steady-run              This is a steady run. Stop it after <brk>
    <cont>convergence

Default
-------
Options common to all PyFoam-applications

--foamVersion=FOAMVERSION
                          Change the OpenFOAM-version that is to be <brk>
                              <cont>used
--currentFoamVersion      Use the current OpenFOAM-version 2.2.x
--force-32bit             Forces the usage of a 32-bit-version if that<brk>
    <cont> version
                          exists as 32 and 64 bit. Only used when --<brk>
                              <cont>foamVersion
                          is used
```

"Documentation" of the PyFoam-utilities

- Finding out what is available
  - All utility-names start with `pyFoam`
    - This is sometimes obnoxious but Tab-complete on the shell gives a full list of the available commands
    - Also makes it unlikely that the name will "clash" with the name from some other package
- Finding out what it does
  - Almost every PyFoam-utility has a `--help`-option
    - Every available option is documented there
- You don't have to write the whole option
  - If the start is unique then the first few characters are sufficient
  - But of course the full option is more "self-documenting"!!"

Running without plotting

- There is an utility that does everything `pyFoamPlotRunner.py` does ... except plotting
  - Useful if you don't need the plots or can't plot (for instance running on a cluster)
- On the terminal start another run

```
pyFoamRunner.py --clear-case --progress  multiphaseInterFoa
```

- Now open another terminal ...

Plotting without running

- On the other terminal go to our case:

```
cd $HOME/PyFoamTraining/damBreakStart
```

- find the logfile and start another utility

```
> pyFoamPlotWatcher.py PyFoamRunner.multiphaseInterFoam.logfile --hardcopy --write-files
GAMGPCG:  Solving for p_rgh, Initial residual = 9.8459e-05, Final residual = 7.79893e-08, No Iterations 4
time step continuity errors : sum local = 1.34994e-06, global = 2.61839e-07, cumulative = 0.000928486
ExecutionTime = 121.99 s  ClockTime = 122 s

Courant Number mean: 0.0521757 max: 0.393277
Interface Courant Number mean: 0.0108982 max: 0.30614
deltaT = 0.00714286
Time = 5.58571

MULES: Solving for alphawater
water volume fraction, min, max = 0.0857768 0 0.994996
...
```

What `pyFoamPlotWatcher.py` does

- Reads the file and does the same analysis that `PyFoamPlotRunner.py` does
- Creates a new directory `<logfilename>.analyzed` with the data
- Waits for the file to change so that it can analyze some more
  - Never-ending: has to be interrupted by the user
- Possible uses:
  - Analyzing the output of a non-PyFoam-controlled run (for instance on the Cluster)
  - Reproduce plots

Watcher vs Redo

- We learned of 2 ways to reproduce plots so far
  - Each has its uses
- pyFoamRedoPlot.py ::
  - Faster because it doesn't have to analyze the log-file
  - Nicer plots because of `matplotlib`
- pyFoamPlotWatcher.py ::
  - Can be used for cases where no PyFoam-data is available
  - Is not restricted to data that PyFoam already analyzed

Connecting to a run over the network

This is advanced stuff. Experiment with it yourself

- Every `PyFoamRunner` starts a little network server
  - In the first terminal restart the run

```
pyFoamRunner.py --clear --progress multiphaseInterFoam
```

- In the second terminal connect to that server:
  - `localhost` for the local computer, 18000 for the server port (next process may have 18001 etc)

```
pyFoamNetShell.py localhost 18000
```

- There is a little shell. Get a list of the available commands

```
PFNET> help
```

- Get help about a specific command

```
PFNET> help stop
```

- End the run (and write current result)

```
PFNET> stop()
```

- BTW: `touch stop` in the case directory would have stopped the run too

Saving our work

- In our "main" shell we go to the parent-directory

```
cd ..
```

- We pack the "important" stuff and the last timestep into an archive

```
pyFoamPackCase.py damBreakStart --last
```

- Checking what is in there

```
> tar tzf damBreakStart.tgz
damBreakStart/system/controlDict
damBreakStart/system/decomposeParDict
damBreakStart/system/fvSchemes
damBreakStart/system/fvSolution
damBreakStart/system/setFieldsDict
damBreakStart/constant/g
```

Cleaning away unnecessary data

- We decide that nothing interesting happened after $t = 2$

```
pyFoamClearCase.py --after=2 damBreakStart
```

- This is much faster and less error prone than doing it "by hand" with commands like

```
rm -r damBreakStart/2.* damBreakStart/3*
```

- As we've seen before the `Runner`-utilities offer the possibility to quickly clean before running with `--clear`

Making a copy the data

- Necessary stuff to run a case (but not all timesteps and other written data) can be easily copied:

```
> pyFoamCloneCase.py damBreakStart <brk>
<cont>damBreakSecondTry --add-item=2
```

- Here we copied an additional timestep too
- It is also possible to create something that looks like a copy but in fact is a reference
  - a new directory structure is created
  - files are symbolic links to the original case

```
> pyFoamCloneCase.py --symlink-mode --relative -<brk>
<cont>symlink damBreakStart damBreakParallel
```

- This is for instance handy when testing two solvers in parallel but we want to make sure the files are consistent

`ls` on Steroids

- `ls` is not very useful for OpenFOAM-cases:

```
> ls
damBreakSecondTry/      damBreakParallel/       damBreakStart/
> pyFoamListCases.py .
                    mtime | first - last (nrSteps) nowTime s    state<brk>
                       <cont> | name
-------------------------------------------------------------------------------<brk>
    <cont>
Tue May 21 23:59:32 2013 |     0 -     2 (      41) 5.58571 s Finished<brk>
    <cont> | ./damBreakStart
Wed May 22 00:11:23 2013 |     0 -     2 (       2)     None s <brk>
    <cont>           | ./damBreakSecondTry
Wed May 22 00:15:51 2013 |     0 -     0 (       1)     None s <brk>
    <cont>           | ./damBreakParallel
```

- This utility only lists directories which are OpenFOAM-cases
  - Sorted by the modification time (newer cases last) but other modes are possible
- Also lists other relevant information like the state and run-time (only available if the case is run by PyFoam) or the written times (available for all cases)
  - Other information like the disk-usage can also be listed
  - For running cases an estimate of when the run will end can be printed

Outline

Preparing a case for parallel running

- We go to the case we already cloned

```
cd damBreakParallel
```

- Remember: this case "only" links to the original

```
> ll system
total 40
controlDict@ -> ../../damBreakStart/system/controlDict
decomposeParDict@ -> ../../damBreakStart/system/decomposeParDict
fvSchemes@ -> ../../damBreakStart/system/fvSchemes
fvSolution@ -> ../../damBreakStart/system/fvSolution
setFieldsDict@ -> ../../damBreakStart/system/setFieldsDict
```

- We want to make sure that we modify a copy, not the original

```
> pyFoamSymlinkToFile.py system/decomposeParDict
> ll system
controlDict@ -> ../../damBreakStart/system/controlDict
decomposeParDict
fvSchemes@ -> ../../damBreakStart/system/fvSchemes
fvSolution@ -> ../../damBreakStart/system/fvSolution
setFieldsDict@ -> ../../damBreakStart/system/setFieldsDict
```

Decomposing the case

```
pyFoamDecompose.py . 3
```

- This call
  1. Creates a new `decomposeParDict` for three processors
     - Using an automatic decomposer (`scotch` or `metis` depending on the OpenFOAM-version)
  2. Runs `decomposePar`
- Other decompositions can be used too

```
> pyFoamDecompose.py . 4 --clear --method=simple --n<brk><cont>=2,2,1 --delta=0.001
```

- This would have failed if the `processorX`-directories were not removed
- The things in newly created directories (here the `processorX`-directories) are **not** symlinks
  - They won't show up in the original case

Running in parallel

- Now we start the case again ... in parallel

```
pyFoamRunner.py --autosense-parallel multiphaseInterFoam
```

- This call
  1. Finds out for how many CPUs the case was decomposed
     - But this can also be specified "by hand" (also a `machinefile` can be specified)
  2. Adds `mpirun` with the correct options (`-np 4` in our case) in front
  3. `-parallel` to tell the solver that this is a parallel run
- If `--autosense` doesn't find `processor`-directories it will run the case in serial

"But we use another MPI-implementation"

- There are multiple points were a simple `mpirun` won't work
  - Because it is not called `mpirun`
  - Because additional options are needed
  - Because different OpenFOAM-versions were compiled against different MPI-implementations
- All these issues can be addressed by the PyFoam-Configuration-system

Format of the configuration files

- The format of the configuration files is similar to the Microsoft Windows INI-files (remember those?)
  - Section names are in []
  - Setting names are on the left of a :
  - The value of the setting is on the right
- Example of a setting:

```
[SolverOutput]
timeregexp: ^(Time =|Iteration:) (.+)$

[Formats]
destination: blue,bold
question: green,standout
source: red,bold
```

Getting the current values of the settings

- All current settings can be listed at once

```
> pyFoamDumpConfiguration.py
...
[MPI]
options_openmpi_post: ["-x","PATH","-x","LD_LIBRARY_PATH<brk>
    <cont>","-x","WM_PROJECT_DIR","-x","PYTHONPATH","-x","<brk>
    <cont>FOAM_MPI_LIBBIN","-x","MPI_BUFFER_SIZE","-x","<brk>
    <cont>MPI_ARCH_PATH"]
openmpi_add_prefix: False
options_openmpi_pre: ["--mca","pls","rsh","--mca","<brk>
    <cont>pls_rsh_agent","rsh"]
run_openmpi: mpirun
...
```

- only the part relevant for `mpirun` is shown in this example

Where configurations are searched

- Configurations are searched in these locations (higher numbers overrule lower numbers):
  1. Hardcoded in the PyFoam-sources
  2. The file `/etc/pyFoam/pyfoamrc`
  3. Files in directory `/etc/pyFoam/pyfoamrc.d`
  4. The file `~/.pyFoam/pyfoamrc` in the user-directory
  5. Files in the directory `~/.pyFoam/pyfoamrc.d`
- 2+3 allow per machine configurations
- 4+5 per user
- The search path and the files actually used are listed by `pyFoamVersion.py`

Version-dependent configuration

- When looking for a setting PyFoam also takes the currently used OpenFOAM-version into consideration
  - as defined in `WM_PROJECT_VERSION`
- When looking for a section `Foo` and there is also a section `Foo-X.Y` and `X.Y` matches the current version then those values are used
  - "Longest" match wins
- A utility to find the current value for an item

```
> pyFoamTestConfiguration.py MPI run_openmpi
Foam-Version:   2.2.x
Section:        MPI
Option:         run_openmpi
Value:          mpirun
```

Example for a version dependent setting

- Assuming that before OpenFOAM 1.7 we used `oldmpirun` and starting with 1.7 `mpirun`
- Add these settings in the search-path:

```
[MPI]
run_openmpi: mpirun

[MPI-1]
run_openmpi: oldmpirun

[MPI-1.7]
run_openmpi: mpirun
```

- This works because there is no OpenFOAM 1.8

How is the case decomposed?

- There is a utility to get information about cases

```
> pyFoamCaseReport.py --decomposition .

Decomposition
=============

Case is decomposed for 4 processors


=============== ==== ==== ==== ====
CPU             0    1    2    3
--------------- ---- ---- ---- ----
Points          1250 1234 1216 1248
Faces           2349 2293 2284 2348
Cells           575  559  559  575
=============== ==== ==== ==== ====
atmosphere      0    0    23   23
defaultFaces    1150 1118 1118 1150
leftWall        26   0    24   0
lowerWall       22   40   0    0
rightWall       0    25   0    25
=============== ==== ==== ==== ====
```

How are the processors coupled

- Another interesting information is which processor talks to which:

```
> pyFoamCaseReport.py --processor-matrix .

Processor matrix
================

Matrix of processor interactions (faces)

=== == == == ==
CPU 0  1  2  3
=== == == == ==
0   0  26 24 0
1   26 0  0  23
2   24 0  0  25
3   0  23 25 0
=== == == == ==
```

Writing the reports

- pyFoamCaseReport.py has some more reports
- The output is ReSTructured Text. So if you got utilities like rst2pdf, rst2html or rst2odt installed you can quickly generate printable/editable documents from it (just pipe)

```
> pyFoamCaseReport.py --short-bc .

Table of boundary conditions for t = 0
==========

========== ========== ========== ========== <brk>
    <cont>========== ==========
..          atmosphere          defaultFaces leftWall       lowerWall <brk>
    <cont>      rightWall
---------- ---------- ---------- ---------- <brk>
    <cont>---------- ----------
Patch Type  patch               empty        wall           wall <brk>
    <cont>          wall
Length      46                  4536         50             62 <brk>
    <cont>            50
========== ========== ========== ========== <brk>
    <cont>========== ==========
U           fluxCorrectedVelocity empty      fixedValue     fixedValue <brk>
    <cont>     fixedValue
alphaair    inletOutlet         empty        alphaContactAngle <brk>
    <cont>alphaContactAngle alphaContactAngle
alphamercury inletOutlet        empty        zeroGradient   zeroGradient <brk>
    <cont>    zeroGradient
alphaoil    inletOutlet         empty        zeroGradient   zeroGradient <brk>
    <cont>    zeroGradient
alphas      zeroGradient        empty        zeroGradient   zeroGradient <brk>
    <cont>    zeroGradient
alphawater  inletOutlet         empty        zeroGradient   zeroGradient <brk>
    <cont>    zeroGradient
p_rgh       totalPressure       empty        fixedFluxPressure <brk>
    <cont>fixedFluxPressure fixedFluxPressure
========== ========== ========== ========== <brk>
    <cont>========== ==========
```

What did change?

- To see what `pyFoamDecompose.py` changed we can use a standard-Unix utility:

```
> diff system/decomposeParDict ../damBreakStart/system/<brk>
    <cont>decomposeParDict
12,14c12,14
< method simple;
< numberOfSubdomains 4;
< simpleCoeffs
---
> method scotch;
> numberOfSubdomains 2;
> scotchCoeffs
16,17d15
<     delta 0.001;
<     n (2 2 1);
```

- Disadvantage of this method is that even if entries are **only** reordered they will be shown as different
  - But for OpenFOAM the value is the same

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

A Utility to view differences

- There is a utility that looks for **semantic** not **textual** differences

```
> pyFoamCompareDictionary.py system/decomposeParDict ../<brk>
    <cont>damBreakStart/
Source file /Volumes/Foam/Cases/Scratch4KoreaPyFoam/<brk>
    <cont>damBreakParallel/system/decomposeParDict
>><< decomposeParDict[numberOfSubdomains] : Differs
>>Source:
 4
<<Destination:
 2
>>>> decomposeParDict[simpleCoeffs] : Missing from <brk>
    <cont>destination
 delta 0.001;
n (2 2 1);

>><< decomposeParDict[method] : Differs
>>Source:
 simple
<<Destination:
 scotch
<<<< decomposeParDict[scotchCoeffs] : Missing from source
```

pyFoamCompareDictionary.py

- Isn't fooled by reformatting or reordering: it reports the semantic differences
  - And on the terminal even colors them
    - Colors can be configured
- The full path to the second file doesn't have to be specified: it searches for the same name as in the first file
  - If more than two files are specified then the last one must be a directory: equivalences for the other files are searched there and reported
- `pyFoamUpdateDictionary.py` is a utility based on this technology that allows interactively copying differences

Outline

Why additional plots?

- `pyFoamPlotRunnerWatcher.py` already allow a number of plots
  - Basically stuff that is output by 90% of all solvers
  - But sometimes we want more
- PyFoam adds the possibility to add custom plots
  - Based on the output of the solver to the terminal
    - By analyzing each line with regular expressions
  - Data also gets written
    - Last item is always available in the `pickledData`-files which is handy for scripts ("What was the pressure difference in the end?")

Regular expressions

- Regular expressions are very popular for analyzing textual data (pattern matching)
  - For instance in OpenFOAM for flexible boundary conditions
  - Python comes with a library for analyzing them
  - There are slightly different dialects
    - For instance there are slight differences between the regular expressions of Python and OpenFOAM
    - But in 90% of all cases they behave the same
- The following slide gives a quick glance
  - Usually you won't need much more for PyFoam

Regular expressions in 3 minutes

1. Most characters match only themself
   - For instance `ab` matches only the string "ab"
2. The dot (`.`) matches **any** character except a newline
   - Pattern `a..a` matches (among others) `abba`, `aBBa`, `ax!a`
3. The plus + matches the character/pattern before it 1 or more times
   - `a.+a` matches `aba`, `abbbba` but not `aa`
4. * is like + but allows no match too
   - `a.*a` matches `aba`, `abbbba` and also `aa`
5. Parenthesis `()` group characters together. Patterns are numbered. They receive the number by the opening `(`
   - `a((b+)a)` would match `abba` with group 1 being `bba` and group 2 `bb`
6. To match a special character like `+-().|` prefix it with a `\`
   - To match `(aa)` you've got to write `\(aa\)`
   - Other special characters that occur frequently in OpenFOAM-output are `[]\{}`

The customRegexp-file

- If a file customRegexp is found in the case by a Plot-utility it is read
- It is in OpenFOAM-format:
  - a dictionary
  - all entries are dictionaries to
- The name of the entry is used to identify the data (for instance during writing)
- Most frequent entry in the dictionaries are:

expr This is required. A regular expression that a line must match. All groups (enclosed by ()) are interpreted as data and plotted

theTitle String with the title of the plot

titles List of words/strings. The names that the data items will get in the legend

Matching floating point numbers

- The pattern to match **all** floating point numbers with regular expressions is quite complex:
  - Matching the sign
  - Exponential notation versus "normal"
- To make life easier PyFoam introduces a shorthand
  - If it finds the string %f% in a regular expression it replaces it with the correct regular expression
- **This only works in PyFoam**. Everywhere else this string will match %f%

What we're trying to do

- In theory the volume fractions in `multiphaseInterFoam` have a clear upper limit:
  - **1**! A phase can not occupy more than 100 percent of the volume
- In practice (numerics) things are not that simple
- Now we want to investigate how far this case deviates from this limit
  - Let's start with air

Plotting the maximum of air

- Find an appropriate line in the solver output

```
air volume fraction, min, max = 0.750463 -3.00394e-27 1.000
```

- Copy/paste it to customRegexp and replace the numbers by patterns

```
volumeFractionAir {
    expr "air volume fraction, min, max = .+ <brk>
        <cont> .+ (%f%)";
    theTitle "Volume fraction of air";
    titles ( air );
}
```

- Run the case (and write plots in the end):

```
pyFoamPlotRunner.py --clear --hardcopy multiphaseInterFoam
```

Plotting a single line

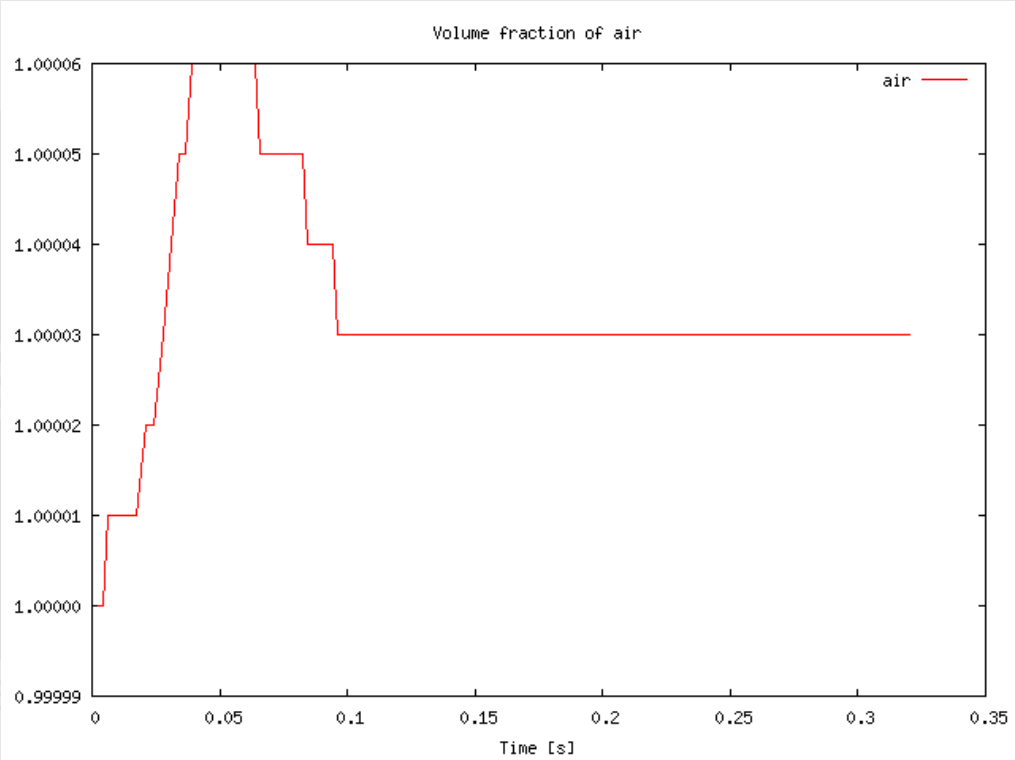


Figure : The file `custom0000.png`

Masters and slaves

- Sometimes we want data from two different lines in one plot
  - But regular expressions can only match one line
- Entries in `customRegexp` don't **have** to open their own plot window
  - They can add their data to another plot (their master)
- To achieve this two additional entries have to be in the dictionary

  type in this case this has to be set to `slave`

  master Name of the master plot
- Here we want to add the sum all species to the plot
  - For this maximum **and** minimum should be 1!

Adding species sum

- Again: find a line

```
Phase-sum volume fraction, min, max = 1 0.999967 1.00003
```

- And append this to the customRegexp

```
volumeFractionSum {
    type slave;
    master volumeFractionAir;
    expr "Phase-sum volume fraction, min, max <brk>
        <cont> = (%f%) (%f%) (%f%)";
    titles ("total average" "total min" "total <brk>
        <cont> max");
}
```

Plot with totals

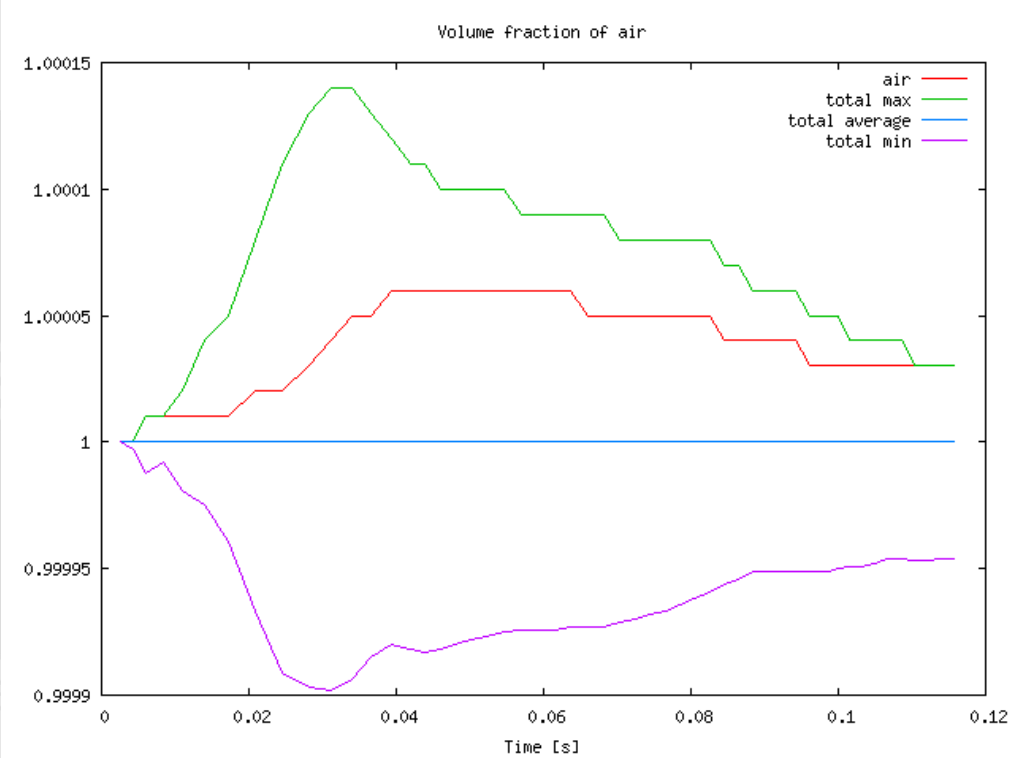


Figure : Four lines (air with total)

Finding bugs in the dictionary

- Sometimes there are problems with `customRegexp`
  - PyFoam issues a warning and then goes on without reading
- There is a utility that reads and interprets a OpenFOAM-dictionary and immediately prints it back

```
pyFoamEchoDictionary.py customRegexp --no-header
```

- Now add an error to the file (for instance remove a {) and see what happens

Dynamic plots

- Now if we want to plot the other phases we could add more slave plots
  - But this is not very elegant
  - Sometimes (for instance with chemical solvers) we don't know beforehand which phases will be there
- To deal with such situations two entries can be added to a dictionary

type Set to `dynamic`. A `dynamic` plot can not be a `slave`

idNr One of the groups in the regular expression will not be interpreted but used as the name of the data set

Plotting the phases

- In customRegexp we replace the entry volumeFractionAir with

```
volumeFraction {
    type dynamic;
    idNr 1;
    expr "(.+) volume fraction, min, max = .+ <brk>
        <cont> .+ (%f%)";
    theTitle "Volume fraction of different <brk>
        <cont>phases";
    titles ( max );
}
```

- Of course in volumeFractionSum the master-entry has to be adapted

Plot with everything together

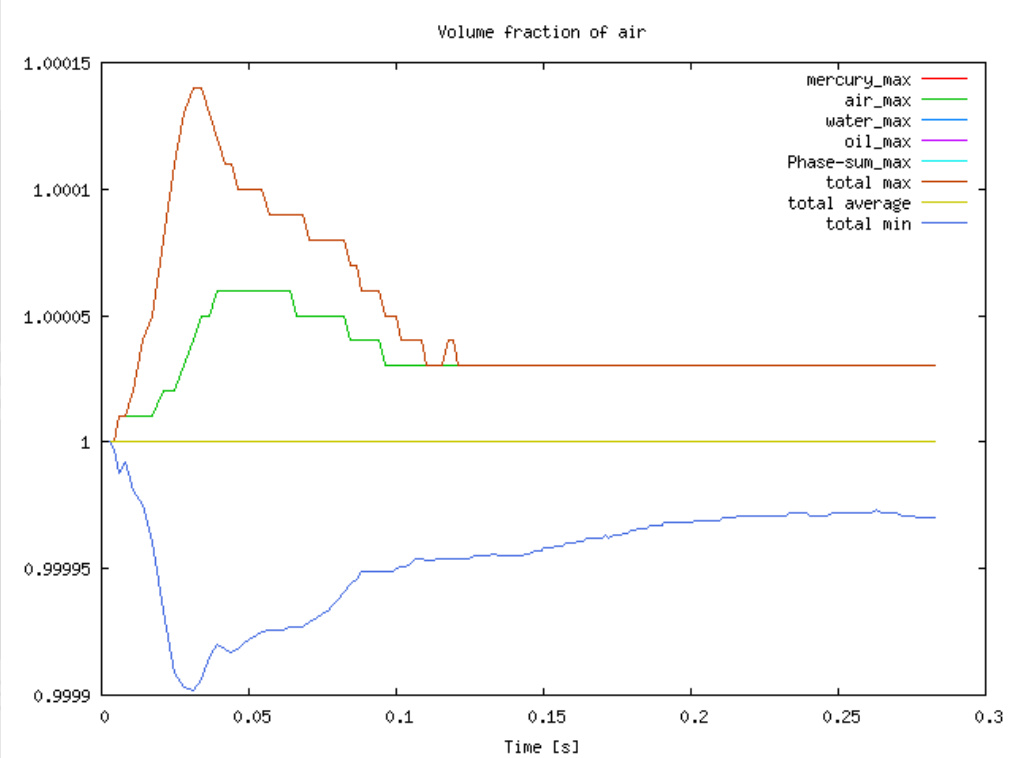


Figure : All phases plus the sum (which is plotted twice)

Wrap-up on the example

- If you don't want `Phase-sum` to show up in the plot change the expression to

```
expr "([^-]+) volume fraction, min, max = .+ .+ (%f%)";
```

- Data is written to the `pickledPlots` and can be exported to `CSV` for further analysis
- Values at the end of the run are available for further analysis:

```
> pyFoamEchoPickledApplicationData.py --pickled-file=Gnuplotting.analyzed/<brk>
    <cont>pickledUnfinishedData --print-data
{'OK': True,
 'analyzed': {'Continuity': {'Cumulative': 8.82218e-05,
                             'Global': 1.74543e-07},
              'Courant': {'max': 0.477438, 'mean': 0.0612366},
              'Custom01_volumeFractionSum': {'total average': 1.0,
                                             'total max': 1.00003,
                                             'total min': 0.999971},
              'Custom02_volumeFraction': {'air_max': 1.00003,
                                          'mercury_max': 1.0,
                                          'oil_max': 1.0,
                                          'water_max': 1.0},
              'DeltaT': {'deltaT': 0.00101507},
              'Execution': {'clock': 0.0, 'cpu': 0.07000000000000028},
              'Iterations': {'Ux': 5.0, 'Uy': 5.0, 'p_rgh': 7.0},
              'Linear': {'Ux': 0.00289564,
                         'Ux_final': 2.10592e-09,
                         'Ux_iterations': 5.0,
                         'Uy': 0.00381379,
                         'Uy_final': 1.0491e-09,
                         'Uy_iterations': 5.0,
                         'p_rgh': 0.00208501,
                         'p_rgh_final': 6.5266e-08,
                         'p_rgh_iterations': 7.0}},
```

Further options in `customRegexp`

- There are some more options that modify the appearance of the plots:

  accumulation if a pattern is found more than once per time-step only the first value is used. This can be changed with this setting (to `last`, `sum` etc)

  logscale make y-scale logarithmic

  with use `points`, `steps` etc instead of `lines`

  alternateAxis A list with names (from `titles`) of data-sets that should be plotted using an alternate y-axis (because their magnitude is different from the others)

  xlabel, ylabel, y2label Labels on the different axes

Outline

Standard function-objects in OpenFOAM

- OpenFOAM has three kinds of functionObjects that generate simpler views into the data
  - **probes** values of certain fields at specified positions as a function of time
  - **sets** values of the fields at a certain time on a 1-dimensional subset of space (usually a line)
  - **surfaces** same as sets but in 2 dimensions (usually a plane)
- PyFoam has utilities that assist the generation of graphs from this

Preparation of the case

- In the current case replace the default `controlDict` with a prepared one

```
> cp ~/IHavePreparedSomething/controlDict.<brk>
  <cont>withInstrumentation system/<brk>
  <cont>controlDict
```

- This adds functionObjects with
  - 3 probes
  - 1 sample line
  - 1 cutting plane through the whole geometry
- Before we go on we run the case to generate the data

```
pyFoamRunner.py --clear --progress multiphaseInterFoam
```

Location of probes and sample-line

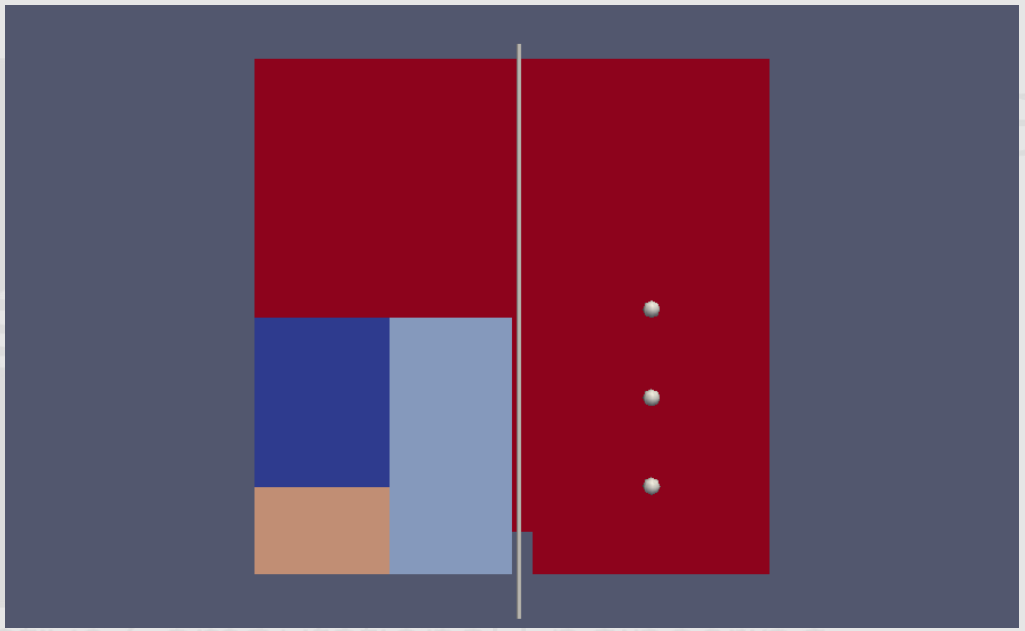


Figure : Points are probes. Grey line is the sample-line

Utilities for probes and samples

- These two utilities work similar
  - Both do not do the plotting themselves: they generate commands for `gnuplot` and print them
    - Actual bitmaps are easily generated by piping the output into `gnuplot`
    - Output can be edited (or processed with a script) before generating the pictures
  - Given the directory where the data resides they analyze the file names to determine the fields, times etc (according to the conventions of OpenFOAM)
    - If fields are missing at certain times they "know"
  - Fields, times etc that are to be plotted can be selected and grouped together (for instance all fields at a position in one plot)

pyFoamTimeLinePlot.py

- This utility is not restricted to probes but also plots data from other function objects that is written as a function of time
- First we find out what is there

```
> pyFoamTimelinePlot.py . --dir=postProcessing<brk>
    <cont>/someProbes --info
Write Times      :  ['0']
Used Time        :  0
Fields           :  ['U', 'alphaair', '<brk>
    <cont>alphamercury', 'alphaoil', 'alphas', <brk>
    <cont>'alphawater', 'p'] Vectors:  ['U']
Positions        :  ['(0.45 0.1 0.01)', '(0.45 <brk>
    <cont>0.2 0.01)', '(0.45 0.3 0.01)']
Time range       :  (0.00119048, 6.0)
```

One plot for every field (at all 3 positions)

```
pyFoamTimelinePlot.py . --dir=postProcessing/<brk>
    <cont>someProbes --collect-lines-by=fields <brk>
    <cont>--basic-mode=lines
set term png nocrop enhanced
set xrange [0.00119048:6]
set output "<brk>
    <cont>postProcessing_someProbes_writeTime_0_Value_U_m
    <cont>.png"
set title "Directory: postProcessing/<brk>
    <cont>someProbes WriteTime: 0 Value: U\\\<brk>
    <cont>_mag"
plot "< tr <./postProcessing/someProbes/0/U -<brk>
    <cont>d '()'" using 1:(sqrt($2*$2+$3*$3+$4*<brk>
    <cont>$4)) title "(0.45 0.1 0.01)" with <brk>
    <cont>lines , "< tr <./postProcessing/<brk>
    <cont>someProbes/0/U -d '()'" using 1:(sqrt<brk>
    <cont>($5*$5+$6*$6+$7*$7)) title "(0.45 0.2<brk>
```

Gnuplot output

- The commands for `gnuplot` does several things
  - Write pictures as `PNG`
  - Automatically generates sensible filenames
  - For vector fields calculates the absolute value
    - Certain components can be selected via options for the utility
  - Adds annotations (title, legends)
- Sending this to a pipe generates some pictures:

```
> pyFoamTimelinePlot.py . --dir=postProcessing <brk>
  <cont>/someProbes --collect-lines-by=fields <brk>
  <cont> --basic-mode=lines | gnuplot
> display <brk>
  <cont>postProcessing_someProbes_writeTime_0_Value <brk>
  <cont>*
```

Plot of oil

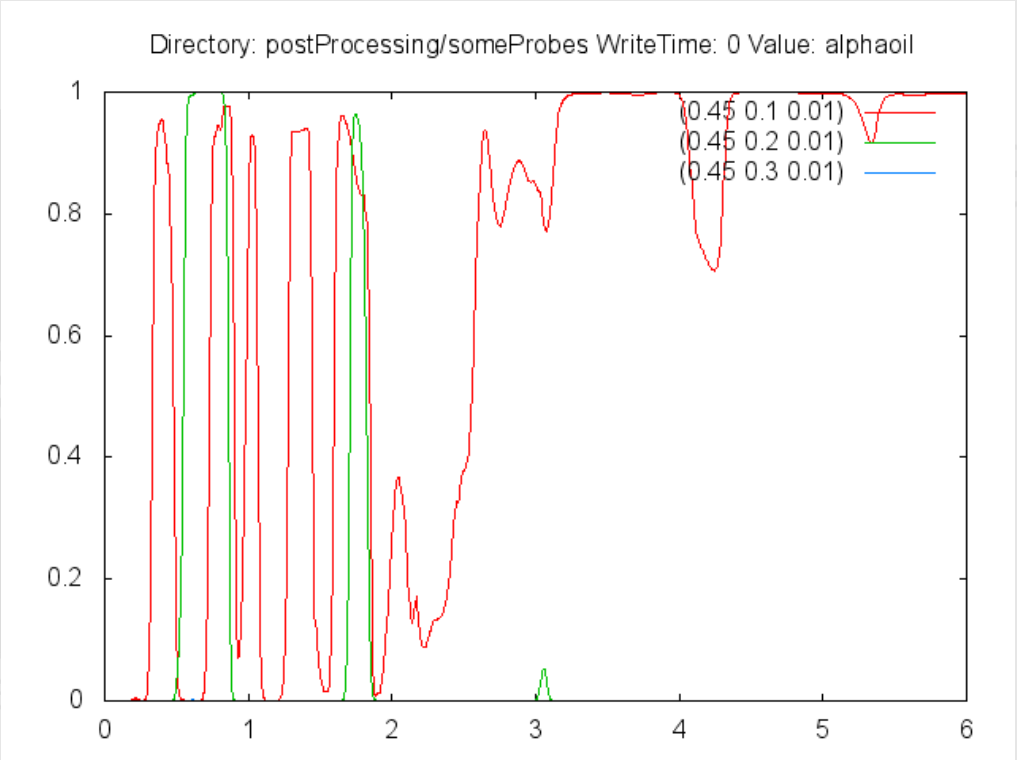


Figure : Fraction of oil at the three locations

Velocities

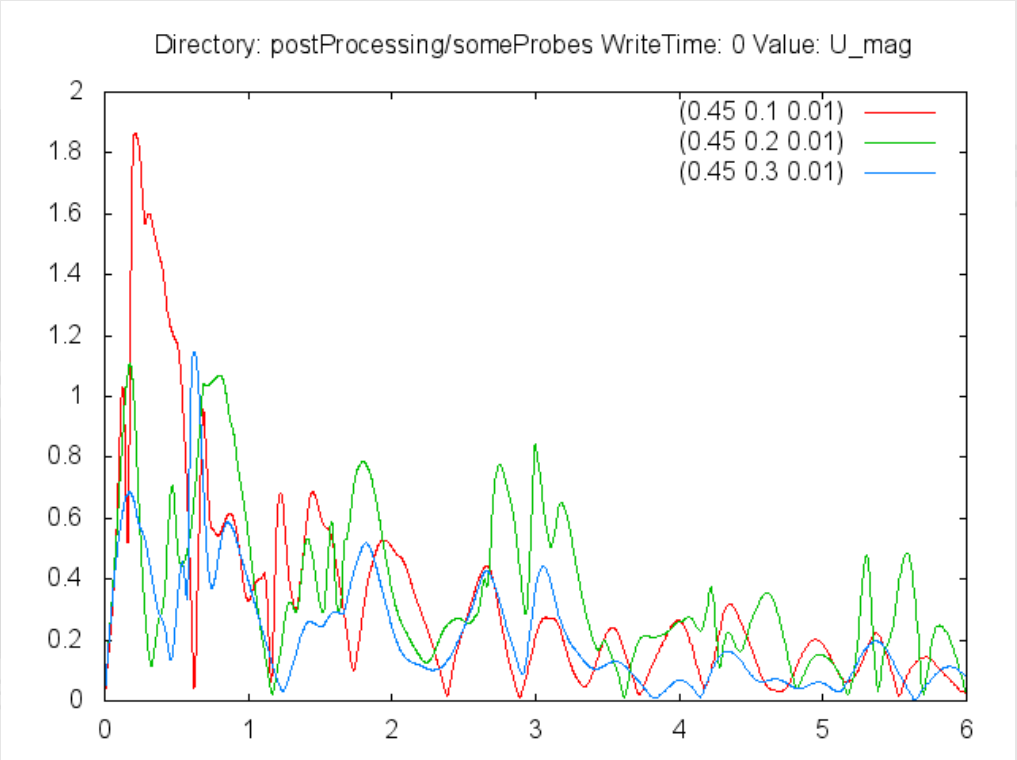


Figure : Absolute velocity at the three locations

Different plots

- Fields at a position

```
> pyFoamTimelinePlot.py . --dir=postProcessing<brk>
  <cont>/someProbes --collect-lines-by=<brk>
  <cont>positions --basic-mode=lines --field=<brk>
  <cont>alphaoil --field=alphamercury --field <brk>
  <cont>=alphawater | gnuplot
```

- Bar plots of values at a certain time

```
1 > pyFoamTimelinePlot.py . --dir=postProcessing<brk>
  <cont>/someProbes --basic-mode=bars --field <brk>
  <cont>=alphaoil --field=alphamercury --<brk>
  <cont>field=alphawater --field=alphaair --<brk>
  <cont>time=2.1 | gnuplot
```

Plot of fields at one place

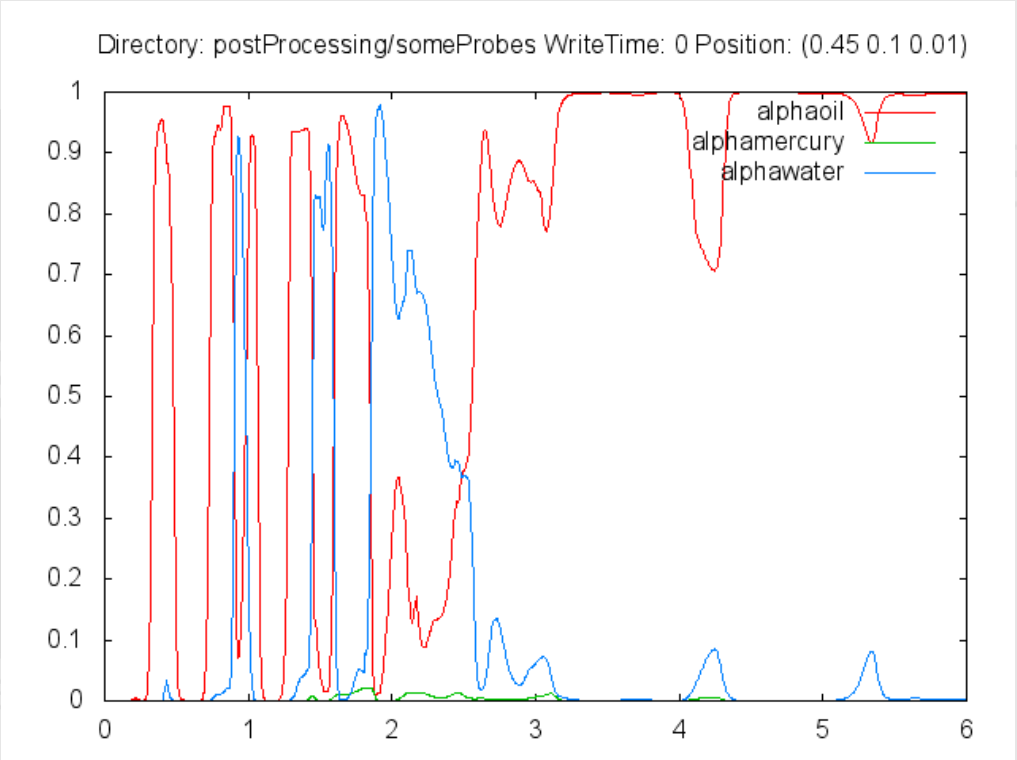


Figure : Fraction of the three heavy liquids at the lowest point

Bar plots of values at one time

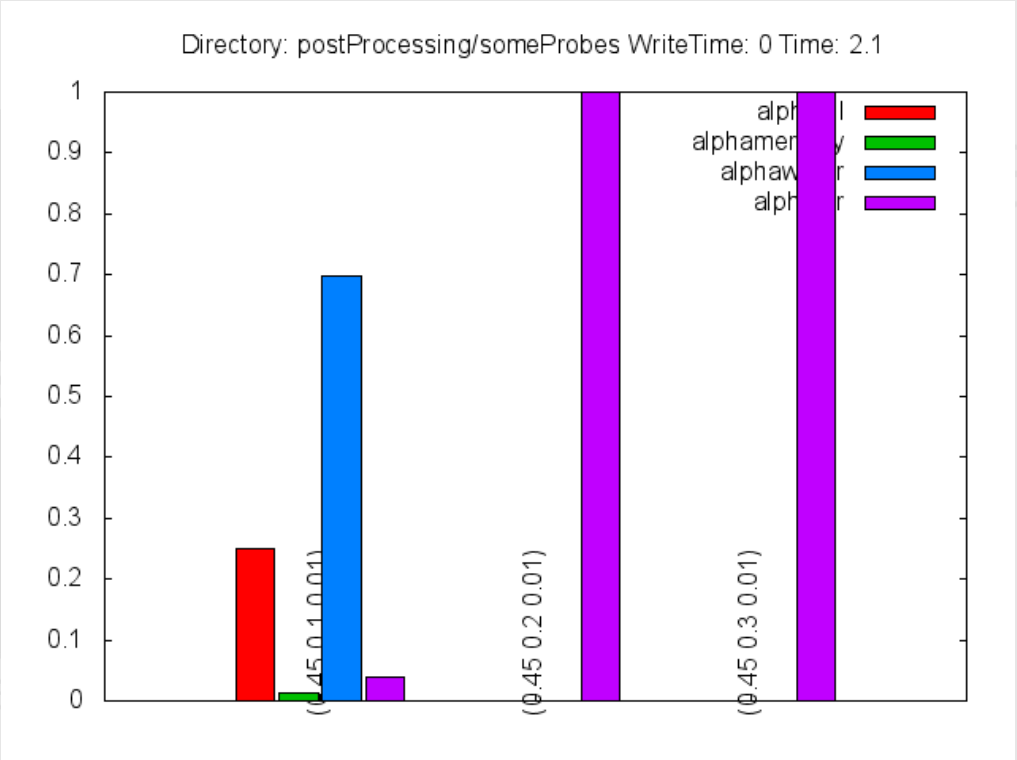


Figure : Bar plots of the liquid fractions

Better living through slicker plots

- If you don't like the way the graphs look (and they **are** not perfect)
  1. Edit the `gnuplot` commands
  2. Write the data to a CSV-file and use program of your choice

```
> pyFoamTimelinePlot.py . --dir=postProcessing/someProbes --collect-lines-by=<brk>
    <cont>positions --basic-mode=lines --field=alphaoil --field=alphamercury --<brk>
    <cont>field=alphawater --csv-file=fractions.csv
> head fractions.csv
time,alphaoil_t=0 (0.45 0.1 0.01),alphaoil_t=0 (0.45 0.2 0.01),alphaoil_t=0 <brk>
    <cont>(0.45 0.3 0.01),alphamercury_t=0 (0.45 0.1 0.01),alphamercury_t=0 (0.45<brk>
    <cont> 0.2 0.01),alphamercury_t=0 (0.45 0.3 0.01),alphawater_t=0 (0.45 0.1 <brk>
    <cont>0.01),alphawater_t=0 (0.45 0.2 0.01),alphawater_t=0 (0.45 0.3 0.01)
1.190480000000000054e-03,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00
2.585029999999999880e-03,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00
4.220029999999999615e-03,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00
6.127529999999999956e-03,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00,0.000000000000000000e<brk>
    <cont>+00,0.000000000000000000e+00
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Quantitative analysis

- The script can also tell you about the properties of the value:

```
> pyFoamTimelinePlot.py . --dir=postProcessing/someProbes --field=<brk>
    <cont>alphamercury --field=alphawater --position="(0.45 0.1 0.01)<brk>
    <cont>" --metrics --basic-mode=lines
Metrics for alphamercury on (0.45 0.1 0.01) index 0 (Path: ./<brk>
    <cont>postProcessing/someProbes/0/alphamercury )
  Min                  : 0.0
  Max                  : 0.0219749
  Average              : 0.00268839895142
  Weighted average     : 0.00238118764677
Data size: 1951
  Time Range           : 0.00119048 6.0

Metrics for alphawater on (0.45 0.1 0.01) index 0 (Path: ./<brk>
    <cont>postProcessing/someProbes/0/alphawater )
  Min                  : 0.0
  Max                  : 0.979092
  Average              : 0.157155334685
  Weighted average     : 0.119943337825
Data size: 1951
  Time Range           : 0.00119048 6.0
```

pyFoamSamplePlot.py

- Plots the output of the `sets`-functionObject
- Before plotting looks through **all** the data and scales for the extremes of **all** times
  - This makes the graphs comparable and allows generating animations from the pictures (y-range will always be the same)
- Also allows you to find out what is there with `--info`

```
> pyFoamSamplePlot.py . --dir=postProcessing/aLine --info
...
> pyFoamSamplePlot.py . --dir=postProcessing/aLine --field=<brk>
  <cont>alphawater --field=alphamercury --field=alphaoil <brk>
  <cont>--mode=fieldsInOne --time=1 --time=3 | gnuplot
> pyFoamSamplePlot.py . --dir=postProcessing/aLine --field=<brk>
  <cont>alphawater --field=alphamercury --field=alphaoil <brk>
  <cont>--mode=timesInOne --time=1 --time=2 --time=3 | <brk>
  <cont>gnuplot
```

Liquids on the line at $t = 1$

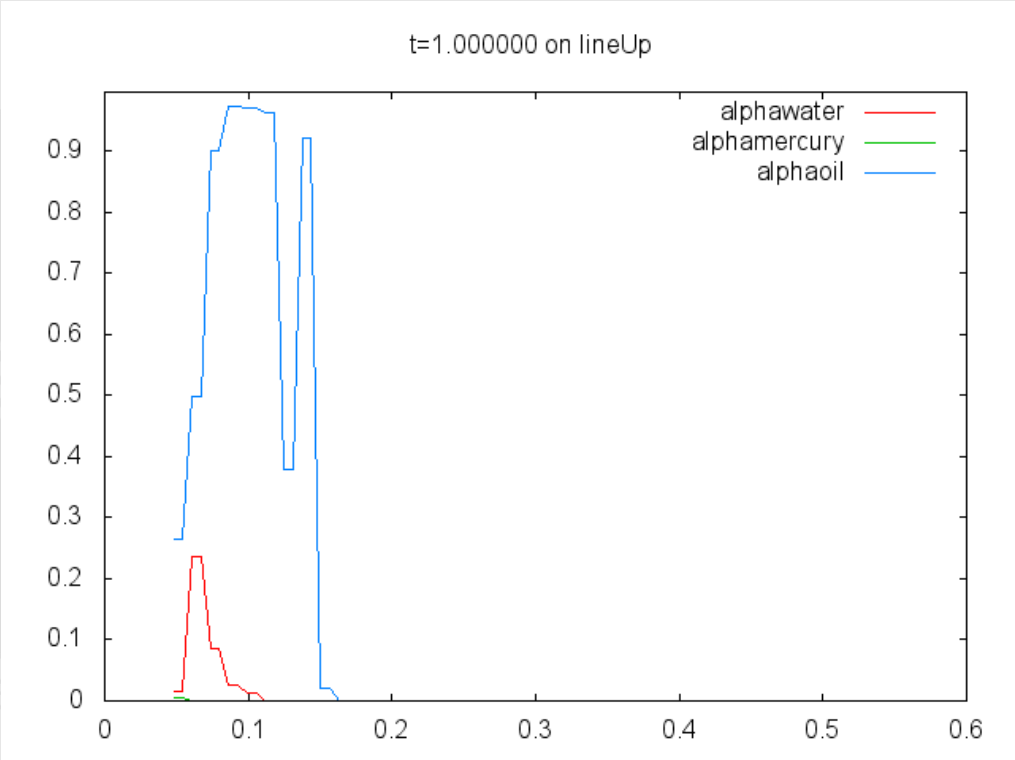


Figure : No liquid reaches 1

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Liquids on the line at $t = 3$

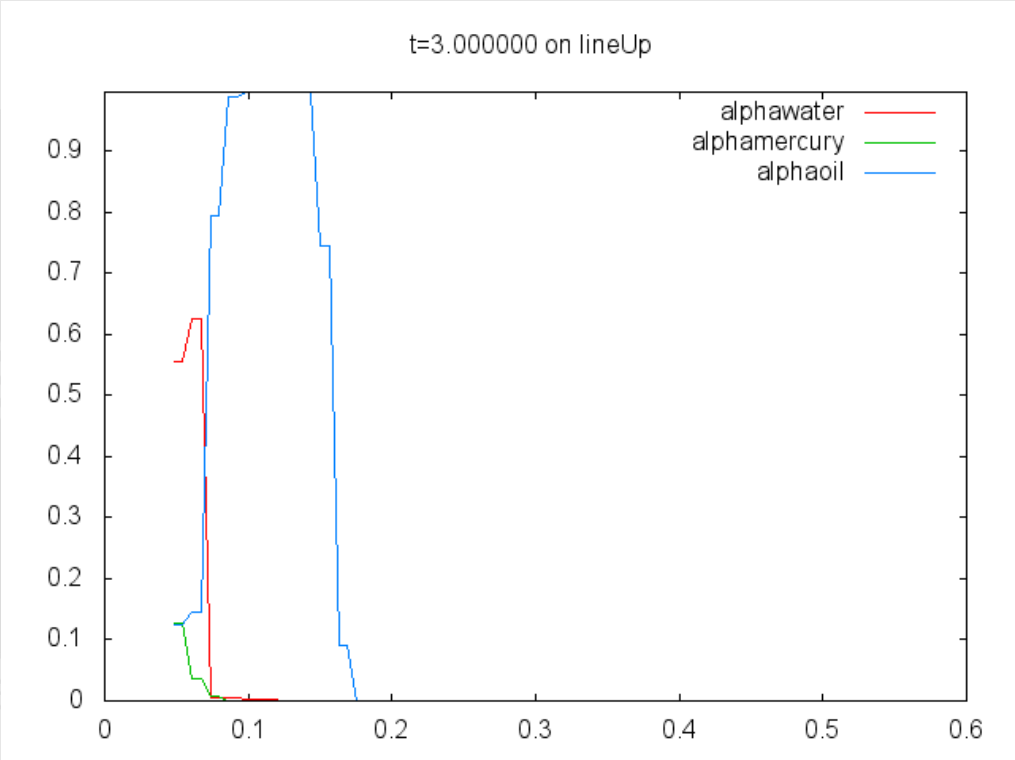


Figure : Now 1 is reached

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Oil at three different times

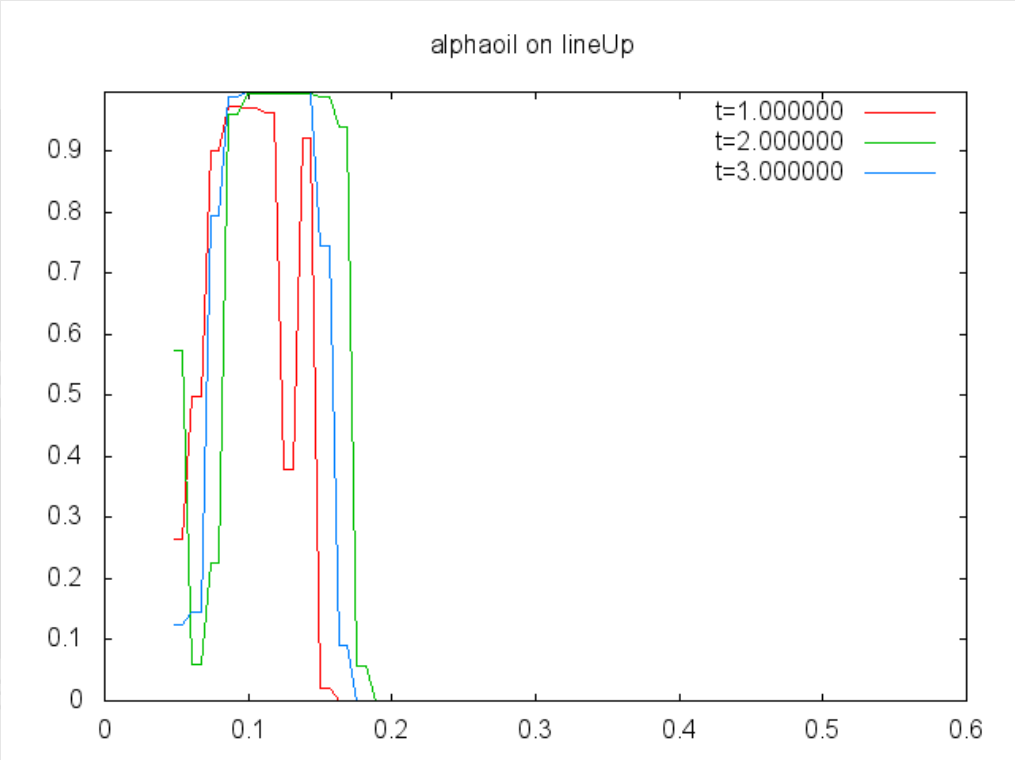


Figure : Evolution of oil

Numbers instead of lines

- Again the analysis of the data

```
> pyFoamSamplePlot.py . --dir=postProcessing/aLine --field=<brk>
  <cont>alphamercury --mode=timesInOne --time=1 --time=2 --time=3 <brk>
  <cont>--metrics
Metrics for alphamercury (Path: ./postProcessing/aLine/1/<brk>
  <cont>lineUp_alphaair_alphamercury_alphaoil_alphas_alphawater_p.<brk>
  <cont>xy )
  Min                : 4.95337e-32
  Max                : 0.00471058
  Average            : 0.000136050755306
  Weighted average   : 0.000109630869949
Data size: 85
  Time Range         : 0.047999 0.584

Metrics for alphamercury (Path: ./postProcessing/aLine/2/<brk>
  <cont>lineUp_alphaair_alphamercury_alphaoil_alphas_alphawater_p.<brk>
  <cont>xy )
  Min                : 1.82851e-24
  Max                : 0.00154467
  Average            : 0.000101984458782
  Weighted average   : 0.000100311458781
Data size: 85
  Time Range         : 0.047999 0.584

Metrics for alphamercury (Path: ./postProcessing/aLine/3/<brk>
  <cont>lineUp_alphaair_alphamercury_alphaoil_alphas_alphawater_p.<brk>
  <cont>xy )
  Min                : 3.80116e-19
  Max                : 0.126732
  Average            : 0.00401707308457
  Weighted average   : 0.00331052841731
Data size: 85
  Time Range         : 0.047999 0.584
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Comparing with reference data

- Both utilities support comparing data with other data sets
  - Plotting
    - When reference data is specified the script tries to find the correct times etc
  - Numerical differences
    - With the `--compare`-option
- The data sets don't have to have the same resolution
  - The script interpolates for numerical comparison
- This allows using data from
  - Different simulations
  - Experiments

pyFoamSurfacePlot.py

- Generates pictures from `surfaces` written in `vtk`-format
  - No need to open `paraview`
  - But with less control
- Needs working Python-bindings for the `VTK`-library
  - This could be a showstopper for older distros
- The utility reads all the date before doing the actual plots. That way
  - it automatically determines the best camera position for the surface
  - it gets the extremes for a field and scale the value ranges for all times to the same value
- It may not produce the most beautiful plots, but for a quick glance it is OK
  - And it is easy to incorporate into a script

Caveat usually `VTK` needs a display to render pictures

Seeing what is there

```
> pyFoamSurfacePlot.py . --dir=postProcessing/<brk>
    <cont>aCut --info
Times      :  ['0.05', '0.1', '0.15', '0.2', <brk>
    <cont>'0.25', ... , '5.75', '5.8', '5.85', <brk>
    <cont>'5.9', '5.95', '6']
Surfaces   :  ['cut']
Values     :  ['U', 'alphaair', 'alphamercury', <brk>
    <cont>'alphaoil', 'alphas', 'alphawater', '<brk>
    <cont>p']
```

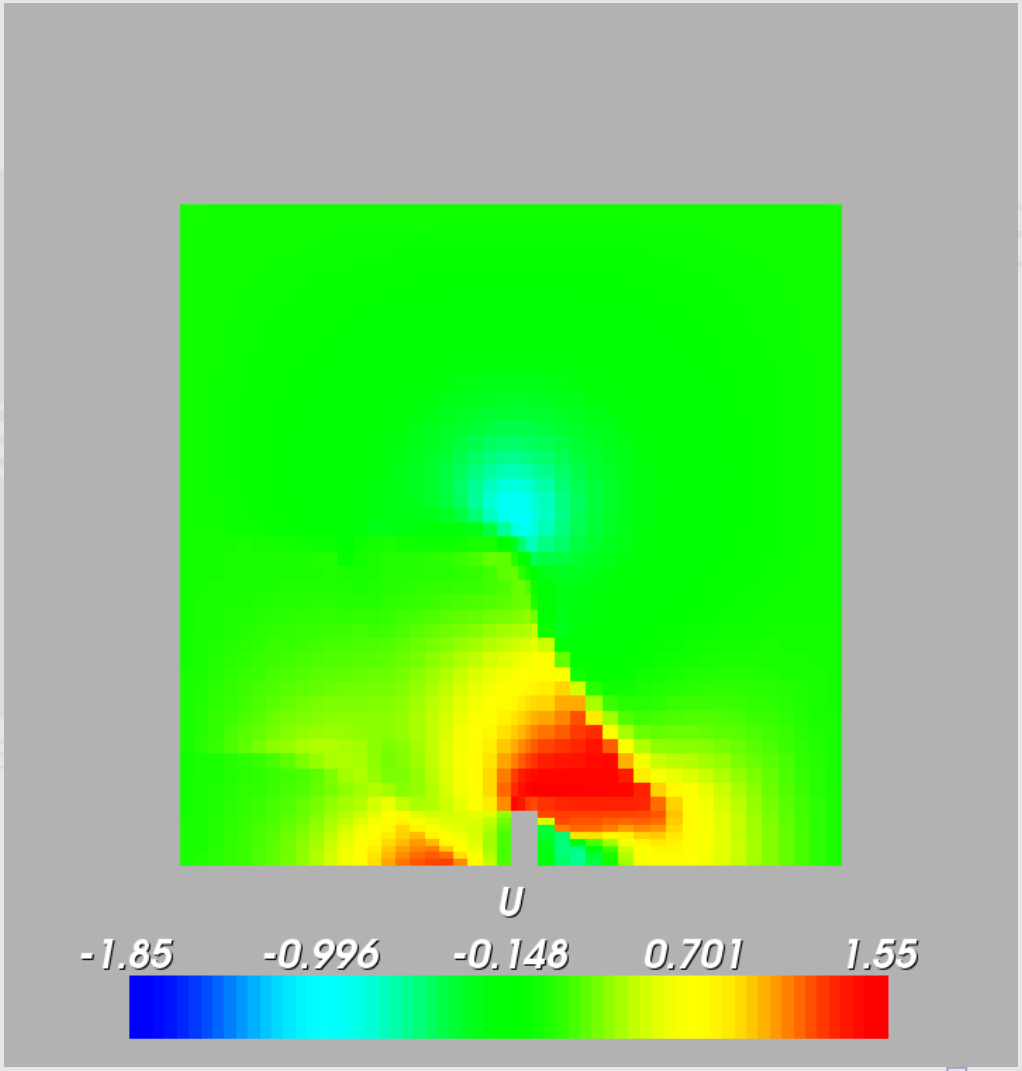
Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Do some actual plotting

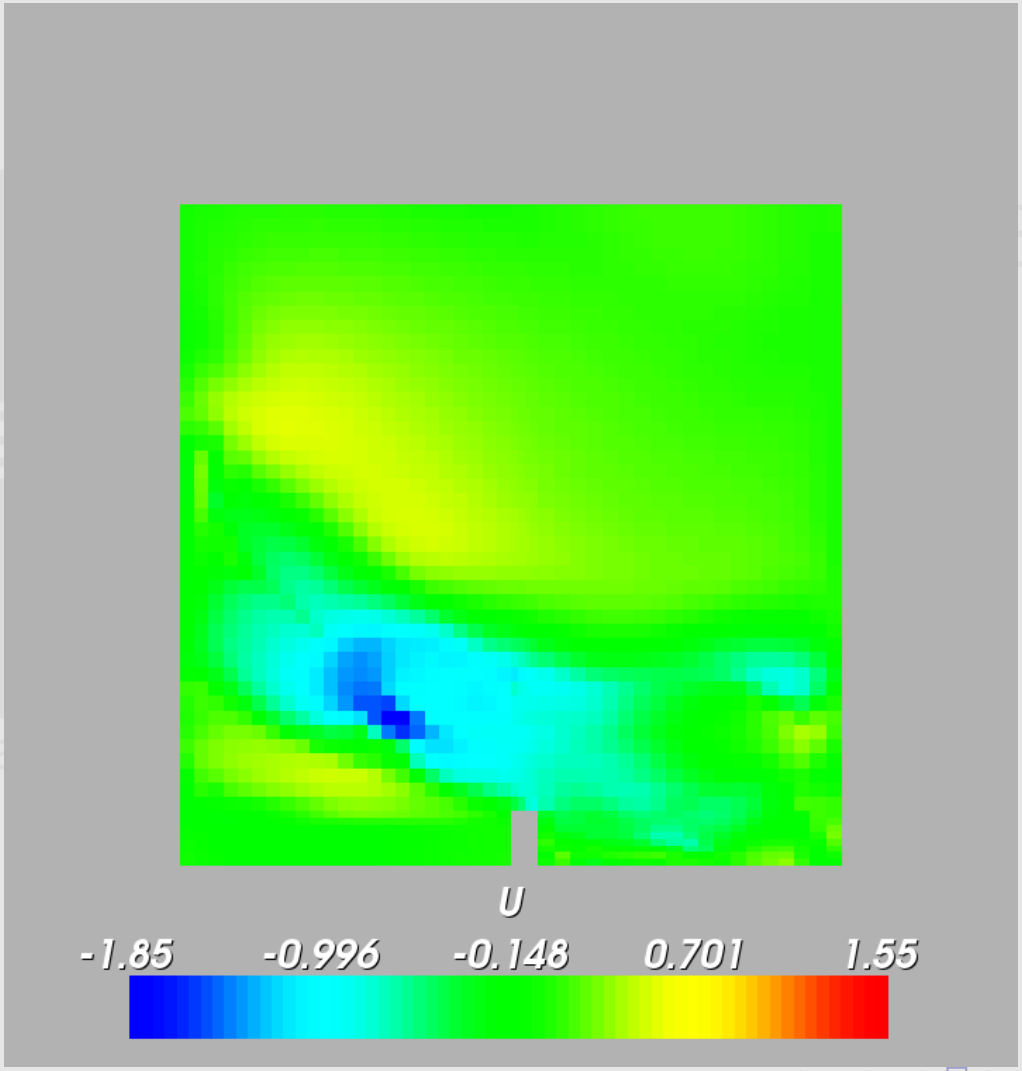
- Plot generates pictures
  - Currently for vectors only the x-component is used
  - Faulty VTK-files give strange results
    - Seems like currently OpenFOAM and VTK disagree on the definition of the VTK-format

```
> pyFoamSurfacePlot.py . --dir=postProcessing/<brk>
    <cont>aCut --time=0.1 --time=1 --time=6 --<brk>
    <cont>field=U --field=alphas
Getting data about plots
Getting ranges
Writing picture postProcessing/aCut_cut_U_0001<brk>
    <cont>.png
Writing picture postProcessing/<brk>
    <cont>aCut_cut_alphas_0001.png
Writing picture postProcessing/aCut_cut_U_0019<brk>
    <cont>.png
Writing picture postProcessing/<brk>
```

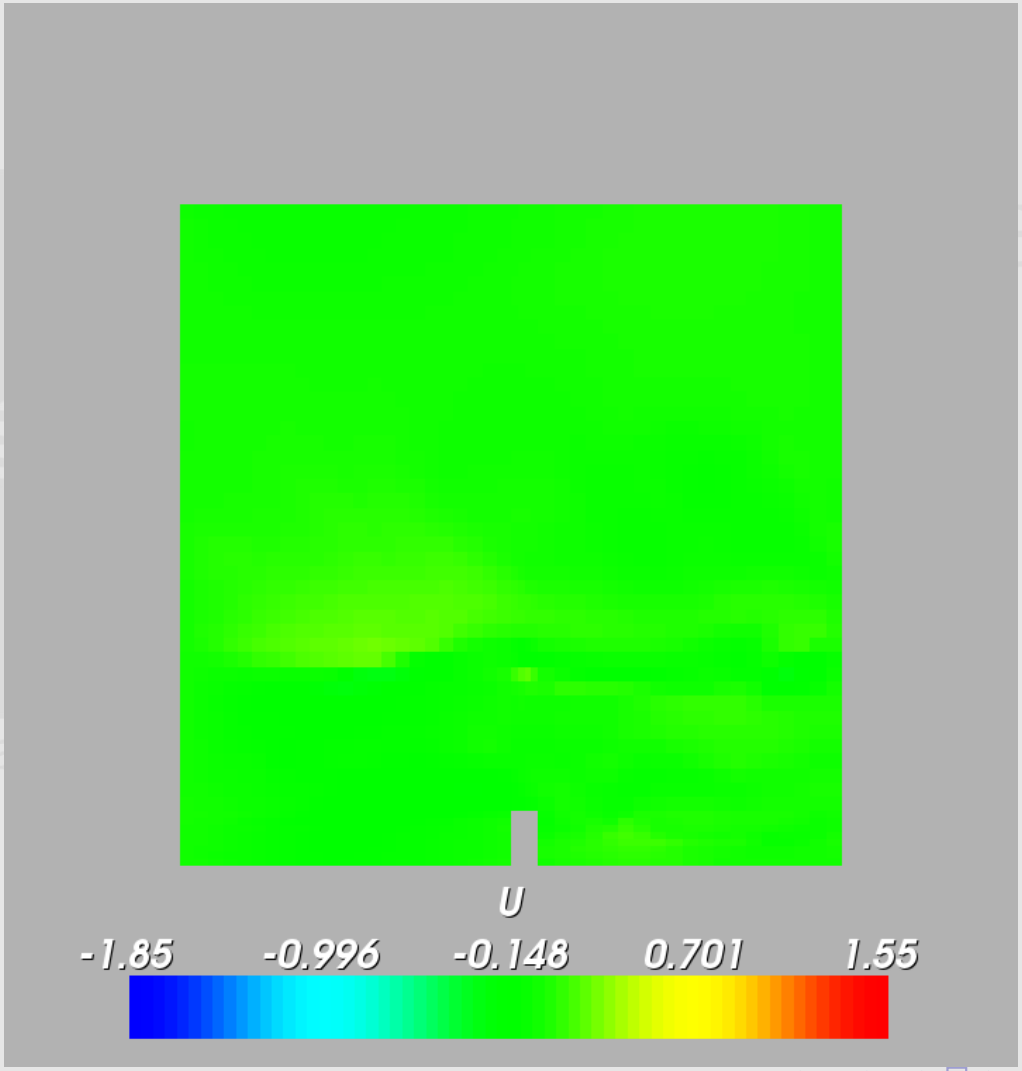
x-Velocity at $t = 0.1$



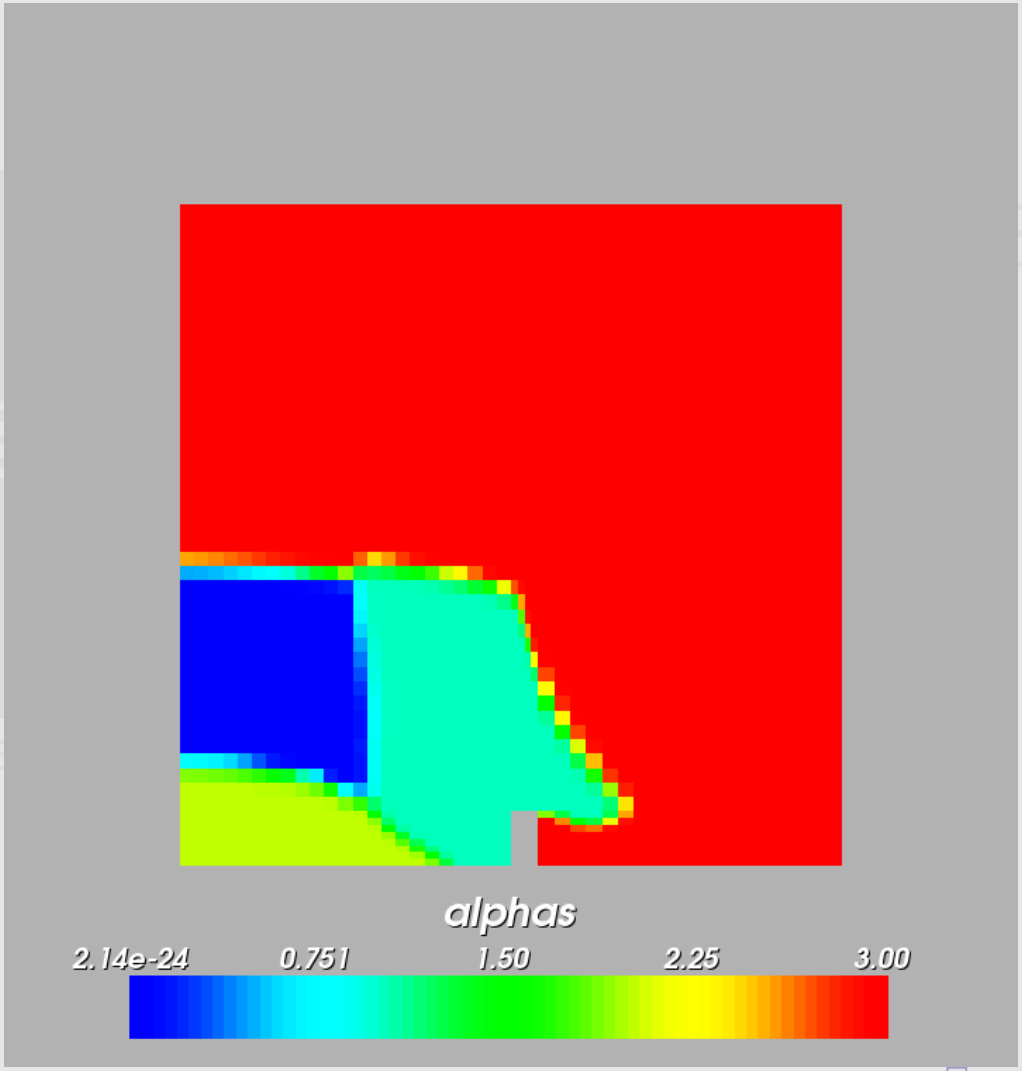
x-Velocity at $t = 1$



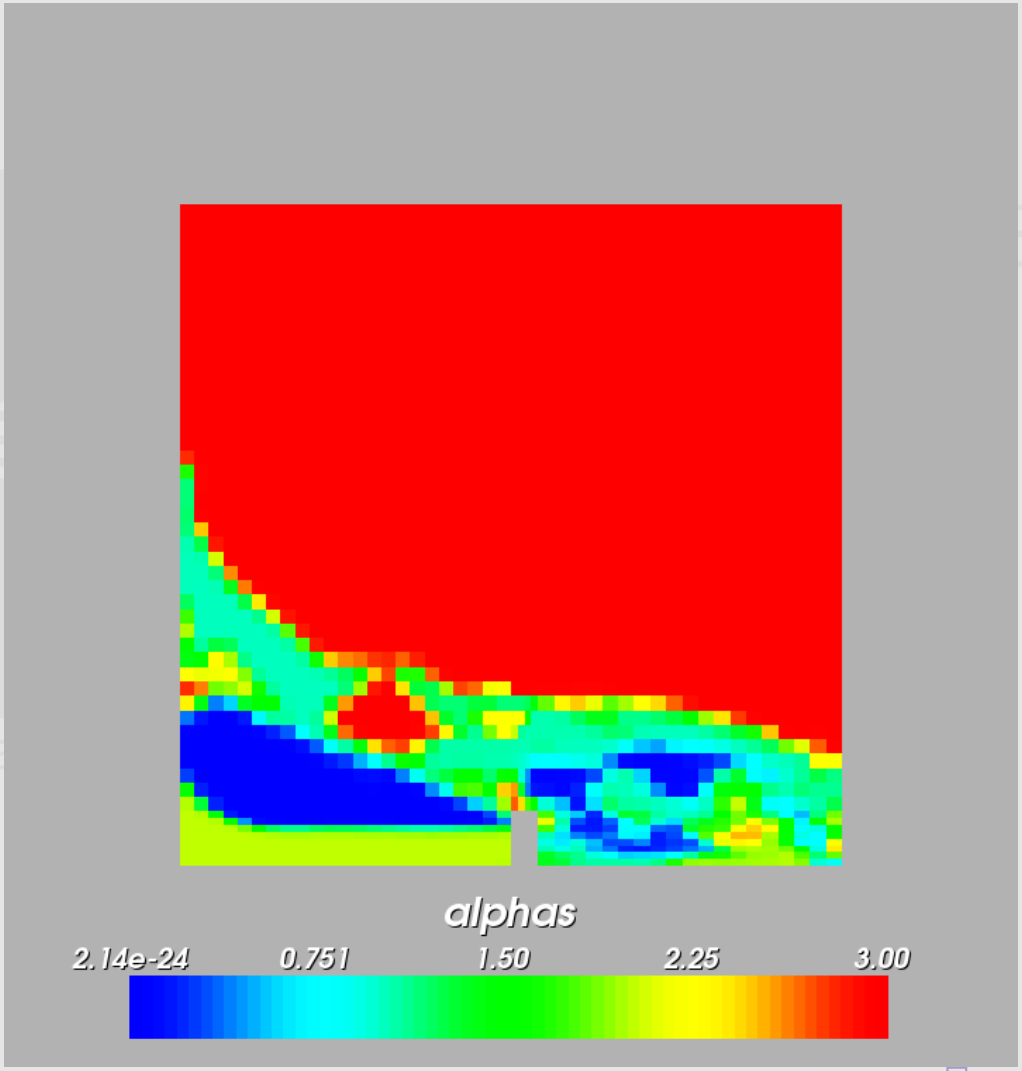
x-Velocity at $t = 6$



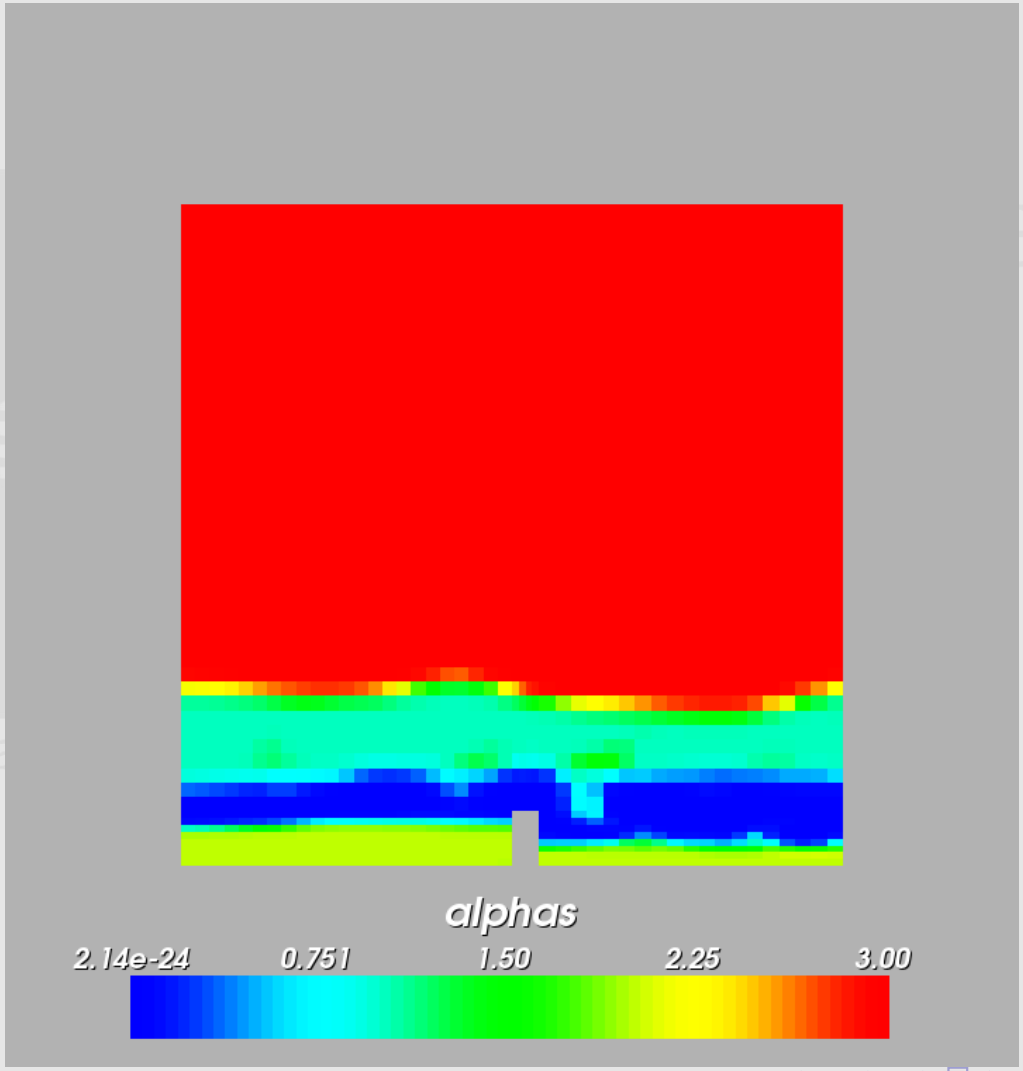
Phases at $t = 0.1$



Phases at $t = 1$



Phases at $t = 6$



Outline

Outline

Ignaz and his colleagues

- Ignaz has to extend the `damBreak`-case in a hurry:
  - "Rebuild the case to simulate spilling tea with milk and honey into a cup"
- and collaborates with two colleagues:

  Isidor Pepranek the local `blockMesh`-guru has to extend the geometry by adding a second wall (to build a cup)

  Irma Pospischil the materials expert is in charge of finding material values for honey (replaces oil), tea (replaces water) and milk (replaces mercury)

  Ignaz himself investigates the differences to the `multiphaseEulerFoam`-solver and whether that one is beneficial

Doing it the old way

- The is the way it is usually done:
  1. Each one starts with his own copy of the case
  2. Makes his own modifications
  3. Afterwards they go through all the files, compare them and copy stuff that is OK
- Problems with that approach
  - It is easy to overlook changes
  - It is not always clear why changes are there
- It would be nice if there was a system to support this . . .

Version control to the rescue

- Actually there are several!
  - They are called Version Control Systems:

    Subversion old school and centralized

    Git Distributed and quite popular

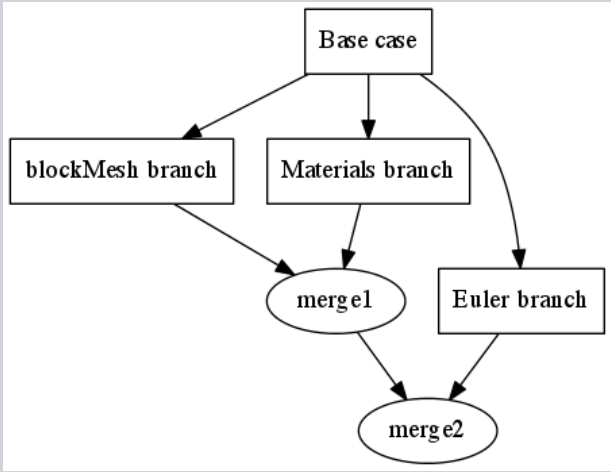
    Mercurial Technically equivalent to `git`. We'll use it here
- These systems are great for tracking changes in text files in directories
  - OpenFOAM-cases are a bunch of text files in directories!!!
- Reason why we use `Mercurial`
  - It is written in Python (and therefor easily programmable in Python)
    - Easily extendable (we'll bring an example for that)
  - It is more straightforward than `git` (I think)

Principles of Distributed Version Control

The principles

- A directory has
  - the current state
  - a database with the whole history
- If changes are OK they are commited to the history
- Directories can be cloned
  - Changes can be pushed to the original directory
- Branches allow splitting different paths of development
  - Can be merged again

Graph of our branches



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Mercurial in 2 minutes

- This is just enough to give you an idea what is happening
- Mercurial commands always start with `hg` and then a word that describes what we really want to do
- Seeing the current changes to the directory

```
hg diff
```

- Committing the changes with a comment

```
hg commit -m "Absolutely essential changes"
```

- Creating a branch

```
hg branch newBranch
```

- Cloning a repository

```
hg clone original copy
```

- Pushing changes to the original

```
hg push
```

- Merge branch into the current one (don't forget to commit afterwards)

```
hg merge newBranch
```

Version control in PyFoam

- PyFoam supports multiple VCS-systems
  - But Mercurial is the default
  - Support for `git` and `svn` is incomplete in some parts
- Utility `pyFoamInitVCSCase.py` initializes a OpenFOAM-case as a repository, adds the most important files and adds some administrative data
- Some utilities are VCS-aware

  pyFoamCloneCase.py if it encounters a case that is under VCS-control it uses clone for that instead of copying files

  pyFoamRunner.py Optionally commits changes before starting a run

Setting up the base case

We start with a fresh case

```
> pyFoamCloneCase.py $BCASE damBreakBase
> cd damBreakBase
> pyFoamInitVCSCase.py .
adding .hgignore
adding constant/g
adding constant/motionProperties
adding constant/polyMesh/blockMeshDict
...
system/fvSolution
system/setFieldsDict
committed changeset 0:dc8a4ee42386
> cp ~/IHavePreparedSomething/controlDict.<brk>
    <cont>withInstrumentation system/controlDict
> cp ~/IHavePreparedSomething/customRegexp .
> hg add customRegexp
adding customRegexp
> hg commit -m "Adding 'standard' evaluations"
customRegexp
system/controlDict
committed changeset 1:f3a3cbf4b8e4
```

Always get a full log of the activities

With a VCS we alway know what was done when:

```
> hg glog
@  changeset:   1:f3a3cbf4b8e4
|  tag:         tip
|  user:        Ignaz Gartengschirrl <igarten@dambreakers.<brk>
    <cont>com>
|  date:        Thu May 23 16:25:27 2013 +0200
|  files:       customRegexp system/controlDict
|  description:
|  Adding 'standard' evaluations
|
|
o  changeset:   0:dc8a4ee42386
   user:        Ignaz Gartengschirrl <igarten@dambreakers.<brk>
       <cont>com>
   date:        Thu May 23 16:20:09 2013 +0200
   files:       .hgignore 0.org/U 0.org/alphaair 0.org/<brk>
       <cont>alphamercury 0.org/alphaoil 0.org/alphas 0.org/<brk>
       <cont>a
   description:
   Initial commit
```

Every line is accounted for

Mercurial keeps track which line was changed at which commit

```
> hg blame system/controlDict
....
0: runTimeModifiable yes;
0:
0: adjustTimeStep   yes;
0:
0: maxCo            0.5;
0: maxAlphaCo       0.5;
0:
0: maxDeltaT        1;
0:
1: libs (
1:     "libsampling.so"
1: );
1:
1: fieldsOfInterest (
1:     U
```

Outline

Setting up Irma's project

```
> cd ..
> pyFoamCloneCase.py damBreakBase damBreakIrma
updating to branch default
resolving manifests
getting .hgignore
...
getting system/fvSolution
getting system/setFieldsDict
22 files updated, 0 files merged, 0 files <brk>
    <cont>removed, 0 files unresolved
> cd damBreakIrma
>  hg branch materials
marked working directory as branch materials
```

Next commit will go to branch materials

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Material data

- Irma googled around and found these material data

Material	Density kg/m^3	Viscosity Pas
Tea (like water)	1000	1e-3
Honey	1400	40
Milk	1035	3e-3

- She adapts the case accordingly and renames the fields
  - We'll do a little shortcut
  - Renaming files has to be done via `hg` (to let it keep track)

Modifying the case

```
> hg mv 0.org/alphamercury 0.org/alphamilk
moving 0.org/alphamercury to 0.org/alphamilk
> hg mv 0.org/alphaoil 0.org/alphahoney
> hg mv 0.org/alphawater 0.org/alphatea
> tar --strip-components 1 -xvzf ~/<brk>
    <cont>IHavePreparedSomething/irmasCase.tgz
```

Now we should have the same case Irma has

Changes done for the transport properties

- Mercurial keeps track of the textual changes

```
> hg diff constant/transportProperties
diff --git a/constant/transportProperties b/constant/<brk>
    <cont>transportProperties
--- a/constant/transportProperties
+++ b/constant/transportProperties
@@ -17,25 +17,25 @@

 phases
 (
-    water
+    tea
     {
         transportModel Newtonian;
         nu nu [ 0 2 -1 0 0 0 0 ] 1e-06;
         rho rho [ 1 -3 0 0 0 0 0 ] 1000;
     }

-    oil
+    honey
     {
         transportModel Newtonian;
-        nu nu [ 0 2 -1 0 0 0 0 ] 1e-06;
-        rho rho [ 1 -3 0 0 0 0 0 ] 500;
+        nu nu [ 0 2 -1 0 0 0 0 ] 28.5714e-3;
+        rho rho [ 1 -3 0 0 0 0 0 ] 1400;
     }
```

Looking at changes the Foam-way

- Mercurial allows the development of extension
  - With PyFoam comes an extension `foamdiff` that shows differences the way `pyFoamCompareDictionary.py` does (assuming the file is a OpenFOAM-file)

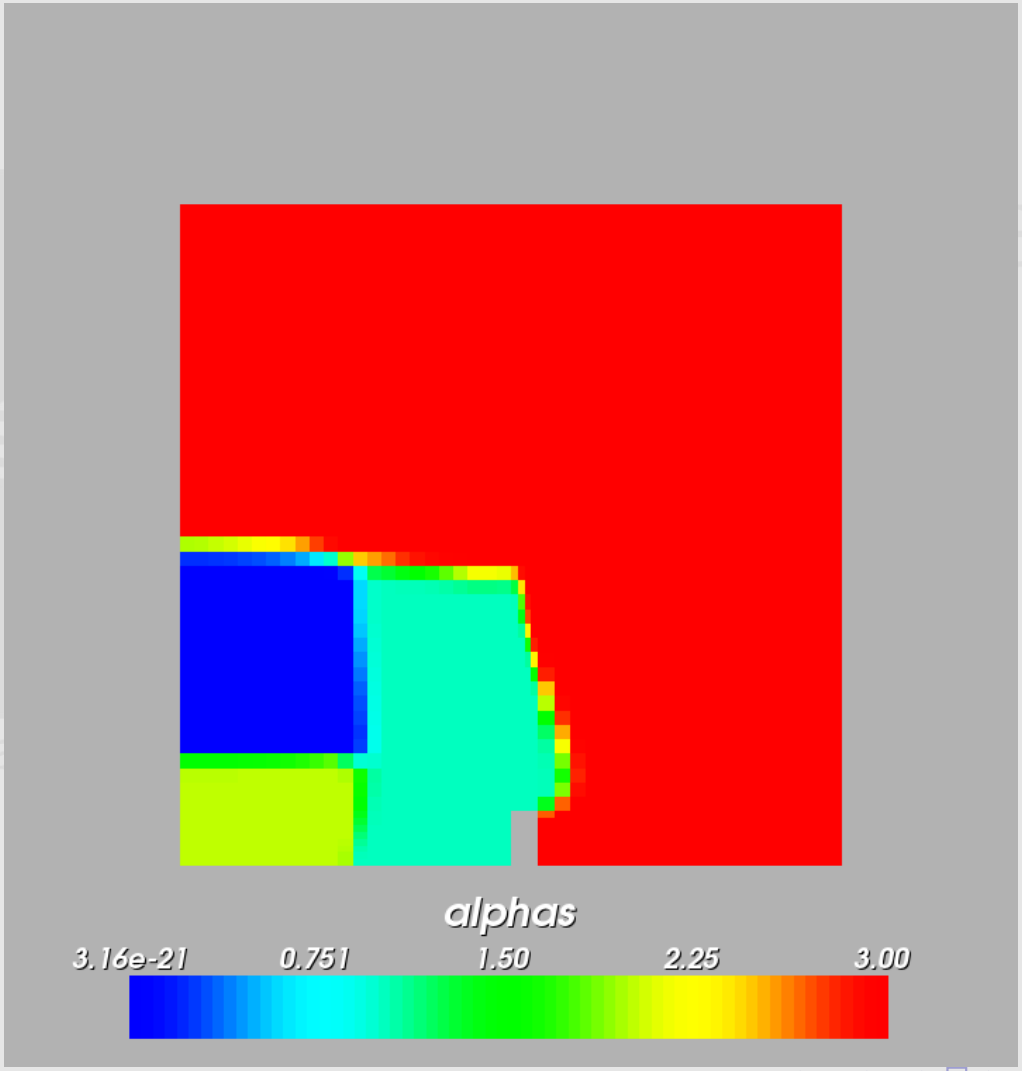
```
> hg foamdiff constant/transportProperties
making snapshot of 1 files from rev f3a3cbf4b8e4
  constant/transportProperties

  Comparing: constant/transportProperties
Source file /var/folders/h7/3nw065_955d1zm30_bjn384h0000gr/<brk>
    <cont>T/foamdiff.jRzJ0q/damBreakIrma.f3a3cbf4b8e4/<brk>
    <cont>constant/transportProperties
>><< transportProperties[phases][0] : Differs
>>Source:
 water
<<Destination:
 tea
>><< transportProperties[phases][2] : Differs
>>Source:
 oil
<<Destination:
 honey
>><< transportProperties[phases][3][nu][2] : Differs
>>Source:
 1e-06
<<Destination:
 0.0285714
>><< transportProperties[phases][3][rho][2] : Differs
>>Source:
 500
<<Destination:
 1400
```

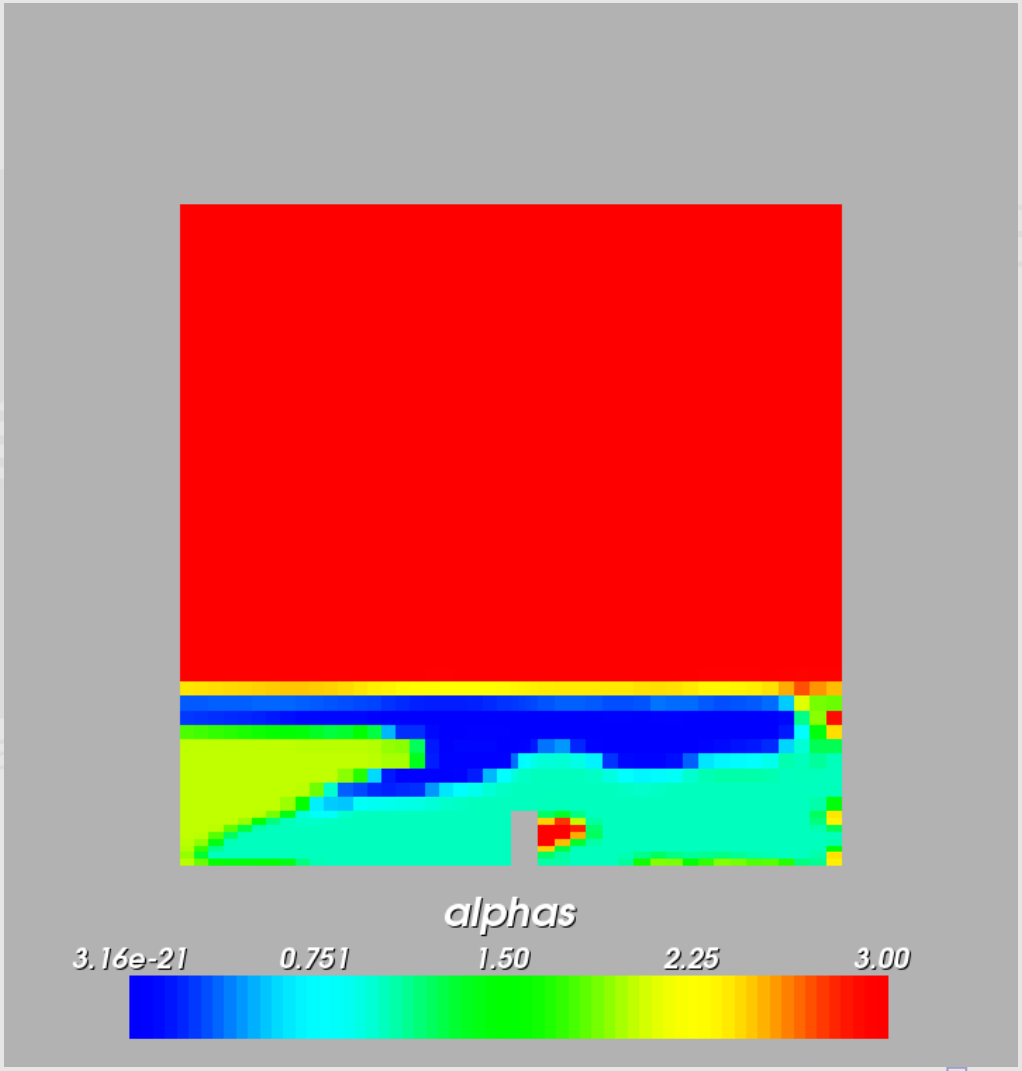
Preparing and running the case

```
> rm -rf 0; cp -r 0.org 0
> blockMesh
...
> setFields
...
> pyFoamPlotRunner.py --clear --commit-to-vcs --message="<brk>
    <cont>New material parameters implemented" <brk>
    <cont>multiphaseInterFoam
 Reading regular expressions from /Volumes/Foam/Cases/<brk>
     <cont>Scratch4KoreaPyFoam/VCS/damBreakIrma/customRegexp
Clearing out old timesteps ....
0.org/alphaair
0.org/alphahoney
0.org/alphamilk
0.org/alphatea
constant/polyMesh/boundary
constant/transportProperties
system/setFieldsDict
committed changeset 2:27e352bb4ba9
/*---------------------------------------------------------------------------*\<brk>
    <cont>
| =========                 | <brk>
    <cont>                                                  |
| \\      /  F ield         | OpenFOAM: The Open Source CFD<brk>
    <cont> Toolbox          |
|  \\    /   O peration     | Version:  2.2.x <brk>
    <cont>                                    |
```

Phases at $t = 0.1$



Phases at $t = 6$



Deviation of the phases from the ideal

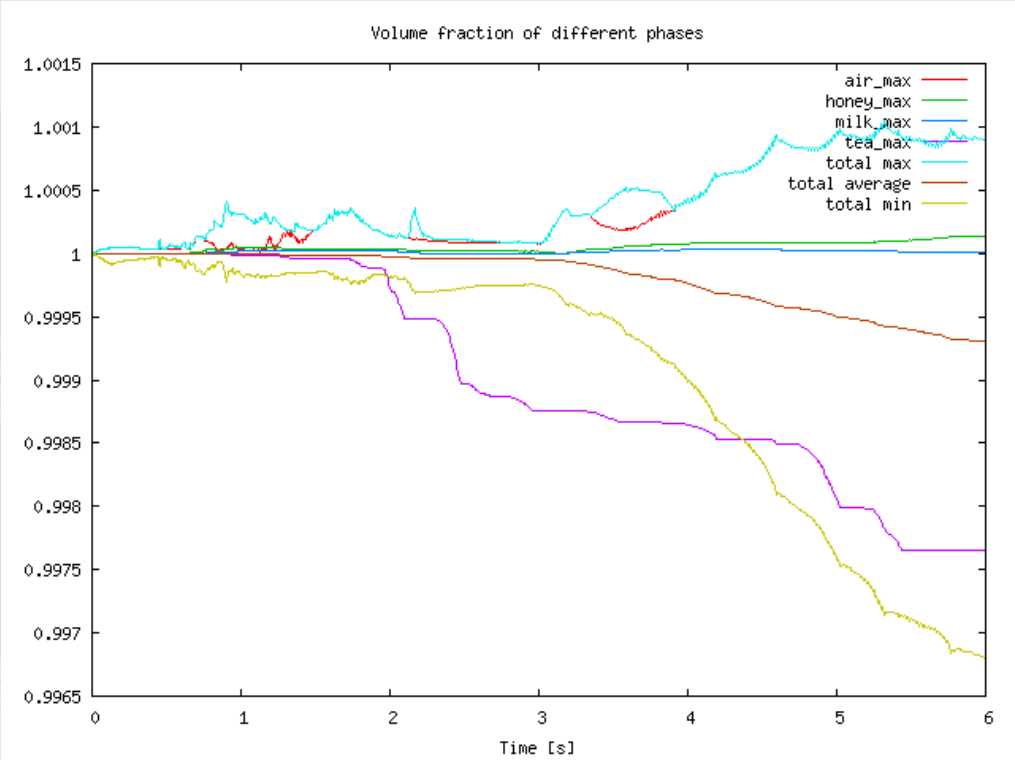


Figure : Difference to 1 is bigger than in the original case

The parameters are not perfect

- The plots show two problems
  - Milk and water do not mix
  - Difference of the phase maxima to 1 is bigger than in the usual case
- Irma decides to work on the parameters.
  - But before she goes on she pushes the changes to the upstream

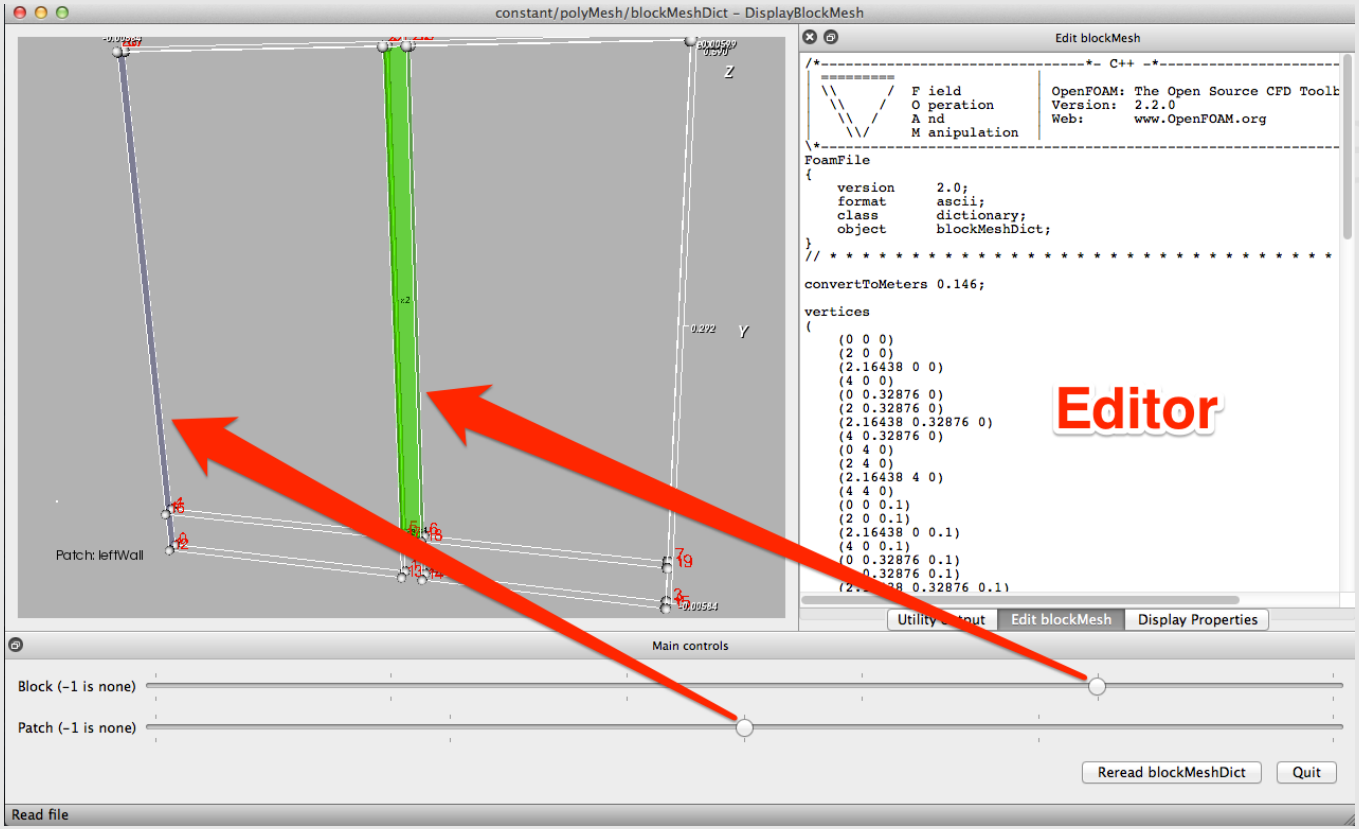
```
> hg push --new-branch
pushing to /Volumes/Foam/Cases/<brk>
    <cont>Scratch4KoreaPyFoam/VCS/damBreakBase
searching for changes
1 changesets found
adding changesets
adding manifests
adding file changes
added 1 changesets with 7 changes to 7 files
running hook changegroup: hg diff --stat -r <brk>
```

Modifying the geometry

- Now we set up Isidors project

```
> cd ..
> pyFoamCloneCase.py damBreakBase damBreakBlock
updating to branch default
resolving manifests
getting .hgignore
getting 0.org/U
...
getting system/fvSolution
getting system/setFieldsDict
22 files updated, 0 files merged, 0 files removed, 0<brk>
    <cont> files unresolved
> cd damBreakBlock
> hg branch templateBlockmesh
marked working directory as branch templateBlockmesh
> pyFoamDisplayBlockMesh.py constant/polyMesh/<brk>
    <cont>blockMeshDict
```

Displaying the blockMeshDict



About `pyFoamDisplayBlockMesh.py`

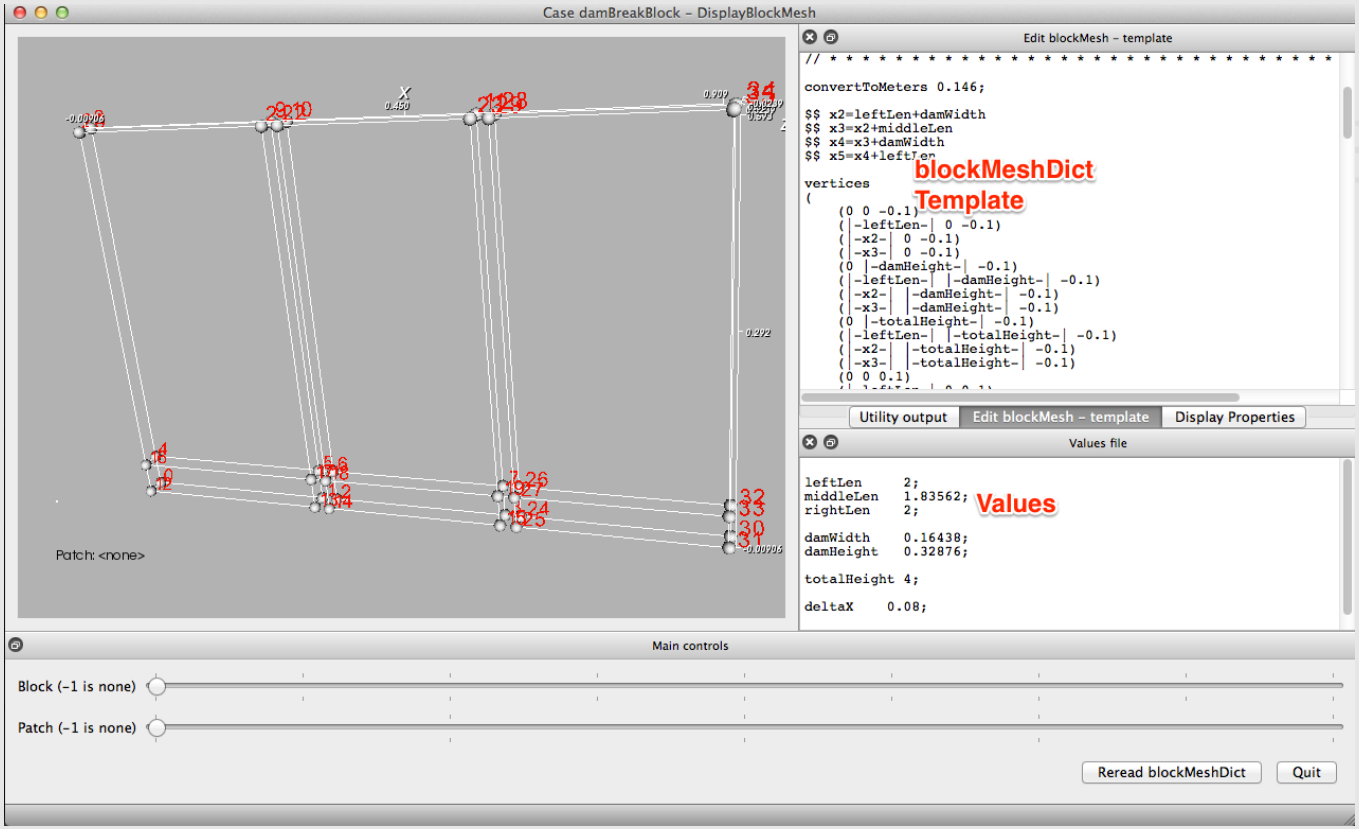
- This utility has the heaviest requirements in terms libraries
  - **PyQt4** Python bindings for the QT-library
  - **vtk** With Python-bindings **and** QT-bindings
- Displays the `blockMeshDict`
  - You can edit it in an external editor
- Allows quickly checking whether patches are correct
- Also allows defining blocks and patches by clicking

Preparing a templated `blockMeshDict`

- Isidor wants to "templatize" the `blockMeshDict` so that later changes are easier
  - Instead of editing we cheat and copy over the files

```
> hg mv constant/polyMesh/blockMeshDict constant/polyMesh/<brk>
    <cont>blockMeshDict.template
moving constant/polyMesh/blockMeshDict to constant/polyMesh<brk>
    <cont>/blockMeshDict.template
> touch constant/polyMesh/meshStandardValues
> hg add constant/polyMesh/meshStandardValues
adding constant/polyMesh/meshStandardValues
> cp ~/IHavePreparedSomething/blockMeshDict.template <brk>
    <cont>constant/polyMesh
> cp ~/IHavePreparedSomething/meshStandardValues constant/<brk>
    <cont>polyMesh
> pyFoamDisplayBlockMesh.py --values-file=constant/polyMesh<brk>
    <cont>/meshStandardValues constant/polyMesh/<brk>
    <cont>blockMeshDict.template
```

Displaying the `blockMeshDict` with a template



Template files in PyFoam

- Template files are text files with special markers
  - Text in the markers is interpreted and replaced with other text
  - The language in which this text is interpreted is ... surprise .... Python
- Markers are (this is the default and can be changed on the command line):
  - $$ on the start of a line. This line is not printed in the final output. The text is an assignment or other Python-expression
  - `|-  -|` : text between this is interpreted as a Python-expression, evaluated, converted to text and then inserted
    - This used to be $ .. $ but then $ became a special character in OpenFOAM-files
- Newer versions of the templates (which are based on the `pyratemp`-engine) allow loops and conditionals
- The utility `pyFoamFromTemplate.py` reads a template file and a file with values and generates a new file

Values file

- The file `meshStandardValues` is a plain OpenFOAM-dictionary with the values
  - Geometric sizes
  - The grid spacing `deltaX`

```
leftLen       2;
middleLen     1.83562;
rightLen      2;

damWidth      0.16438;
damHeight     0.32876;

totalHeight   4;

deltaX        0.08;
```

The template file

- This is only a small part of `blockMeshDict.template`
  - The part that calculates the number of cells from the sizes and `deltaX`

```
$$ nrX1=int(ceil(leftLen/deltaX))
$$ nrX2=int(ceil(middleLen/deltaX))
$$ nrX3=int(ceil(rightLen/deltaX))
$$ nrXDam=int(ceil(2*damWidth/deltaX))
$$ nrYDam=int(ceil(2*damHeight/deltaX))
$$ nrYUp=int(ceil((totalHeight-damHeight)/deltaX))

blocks
(
    hex (0 1 5 4 12 13 17 16) (|-nrX1-| |-nrYDam-| 1) <brk>
        <cont>simpleGrading (1 1 1)
    hex (2 3 7 6 14 15 19 18) (|-nrX2-| |-nrYDam-| 1) <brk>
        <cont>simpleGrading (1 1 1)
    hex (4 5 9 8 16 17 21 20) (|-nrX1-| |-nrYUp-| 1) <brk>
        <cont>simpleGrading (1 1 1)
    hex (5 6 10 9 17 18 22 21) (|-nrXDam-| |-nrYUp-| 1) <brk>
        <cont>simpleGrading (1 1 1)
    hex (6 7 11 10 18 19 23 22) (|-nrX2-| |-nrYUp-| 1) <brk>
        <cont>simpleGrading (1 1 1)
    hex (7 26 28 11 19 27 29 23) (|-nrXDam-| |-nrYUp-| 1) <brk>
        <cont>simpleGrading (1 1 1)
    hex (26 32 34 28 27 33 35 29) (|-nrX3-| |-nrYUp-| 1) <brk>
        <cont>simpleGrading (1 1 1)
    hex (24 30 32 26 25 31 33 27) (|-nrX3-| |-nrYDam-| 1) <brk>
        <cont>simpleGrading (1 1 1)
);
```

Preparing and running the case .. again

```
> rm -rf 0; cp -r 0.org 0
> pyFoamFromTemplate.py --template=constant/<brk>
    <cont>polyMesh/blockMeshDict.template --<brk>
    <cont>output=constant/polyMesh/<brk>
    <cont>blockMeshDict --values-dict=constant/<brk>
    <cont>polyMesh/meshStandardValues
> blockMesh
...
> setFields
...
> hg commit -m "New mesh implemented"
constant/polyMesh/blockMeshDict.template
constant/polyMesh/boundary
constant/polyMesh/meshStandardValues
committed changeset 3:6c6b0bc67056
> pyFoamPlotRunner.py --clear --progress <brk>
    <cont>multiphaseInterFoam
```

Phases at $t = 0.1$

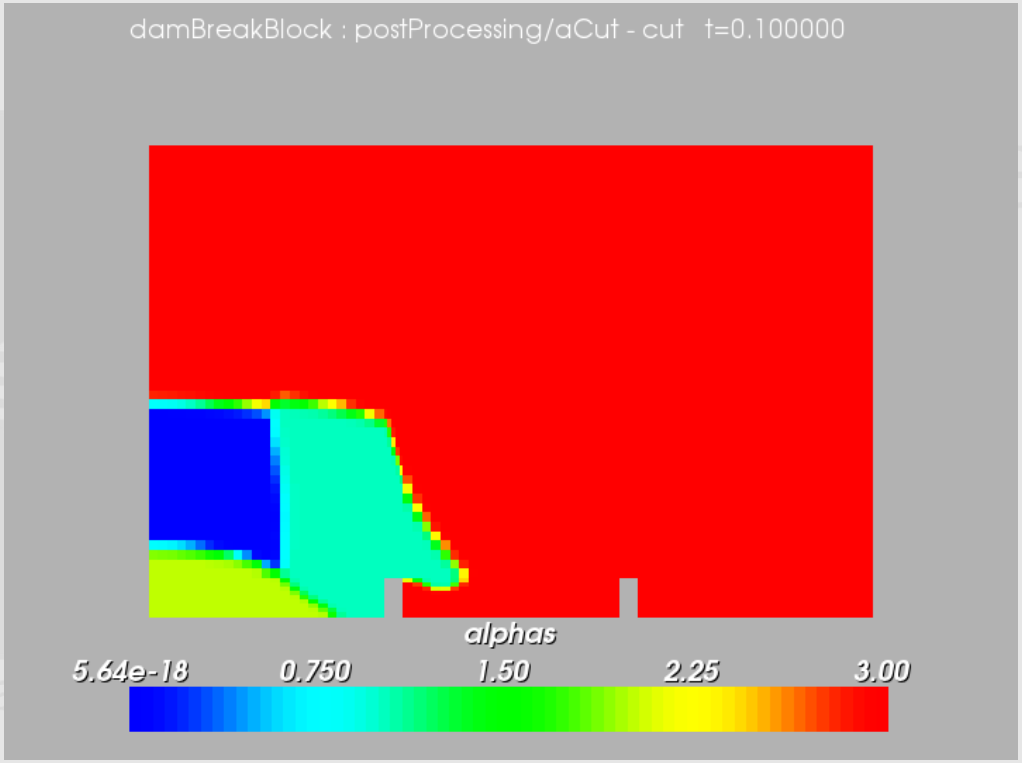


Figure : Looks the same as in the original

Phases at $t = 1$

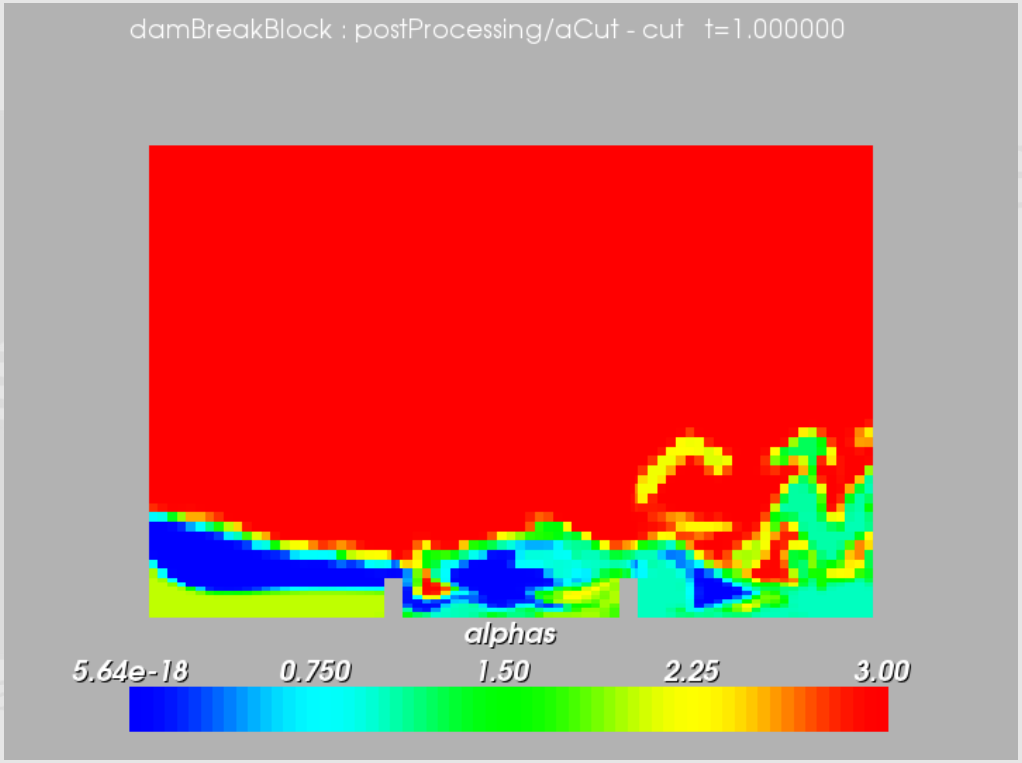


Figure : Splashes over the second dam

Phases at $t = 6$

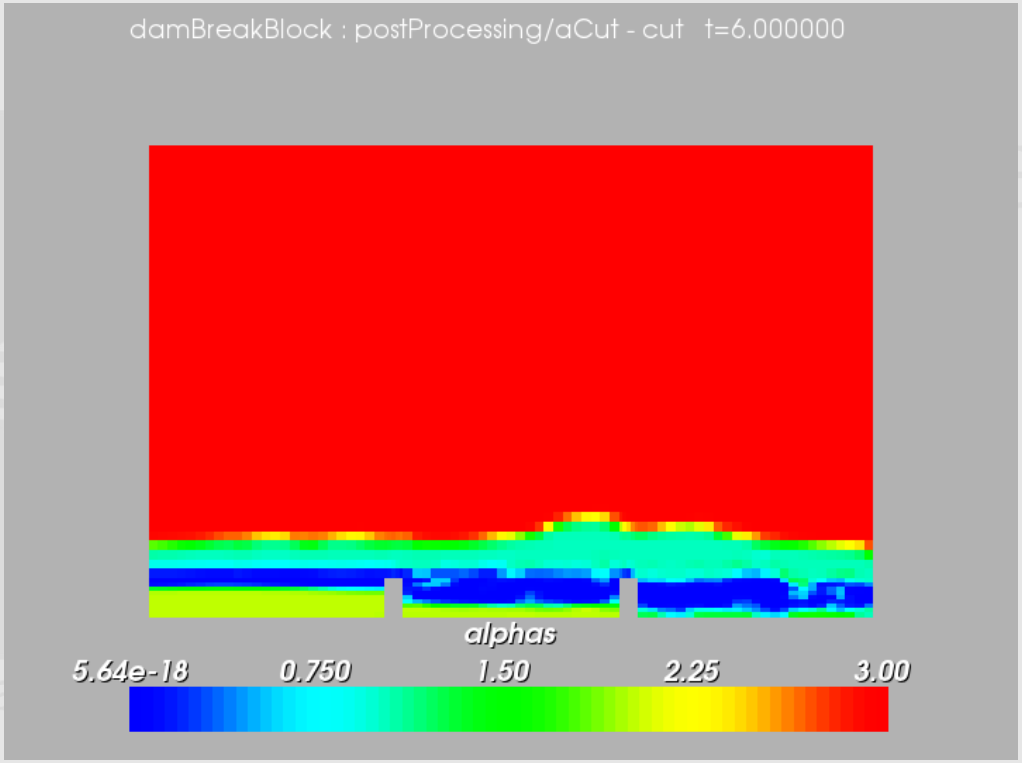


Figure : Separation of the phases

Wrapping up geometry modifications

- Isidor sets a marker, pushes his changes and checks for updates

```
> hg tag meshTemplateFinished
.hgtags
> hg push --new-branch
pushing to /Volumes/Foam/Cases/Scratch4KoreaPyFoam/<brk><cont>VCS/damBreakBase
searching for changes
2 changesets found
adding changesets
adding manifests
adding file changes
added 2 changesets with 4 changes to 4 files (+1 <brk><cont>heads)
running hook changegroup: hg diff --stat -r $HG_NODE<brk><cont> -r tip
 .hgtags |  1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
> hg incoming
comparing with /Volumes/Foam/Cases/<brk><cont>Scratch4KoreaPyFoam/VCS/damBreakBase
searching for changes
no changes found
```

Getting Irmas work

```
> hg branches
templateBlockmesh                4:f51b6b21113e
materials                        2:27e352bb4ba9
default                          1:f3a3cbf4b8e4 (inactive)
> hg merge materials
resolving manifests
removing 0.org/alphamercury
removing 0.org/alphaoil
removing 0.org/alphawater
getting 0.org/alphaair
getting 0.org/alphahoney
getting 0.org/alphamilk
getting 0.org/alphatea
merging constant/polyMesh/boundary
getting constant/transportProperties
getting system/setFieldsDict
6 files updated, 1 files merged, 3 files removed, 0 files unresolved
(branch merge, don't forget to commit)
> hg commit -m "Merge in different materials"
0.org/alphaair
0.org/alphahoney
0.org/alphamilk
0.org/alphatea
constant/polyMesh/boundary
constant/transportProperties
system/setFieldsDict
committed changeset 5:a9a28291f272
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Phases at $t = 0.1$ - The tea variation

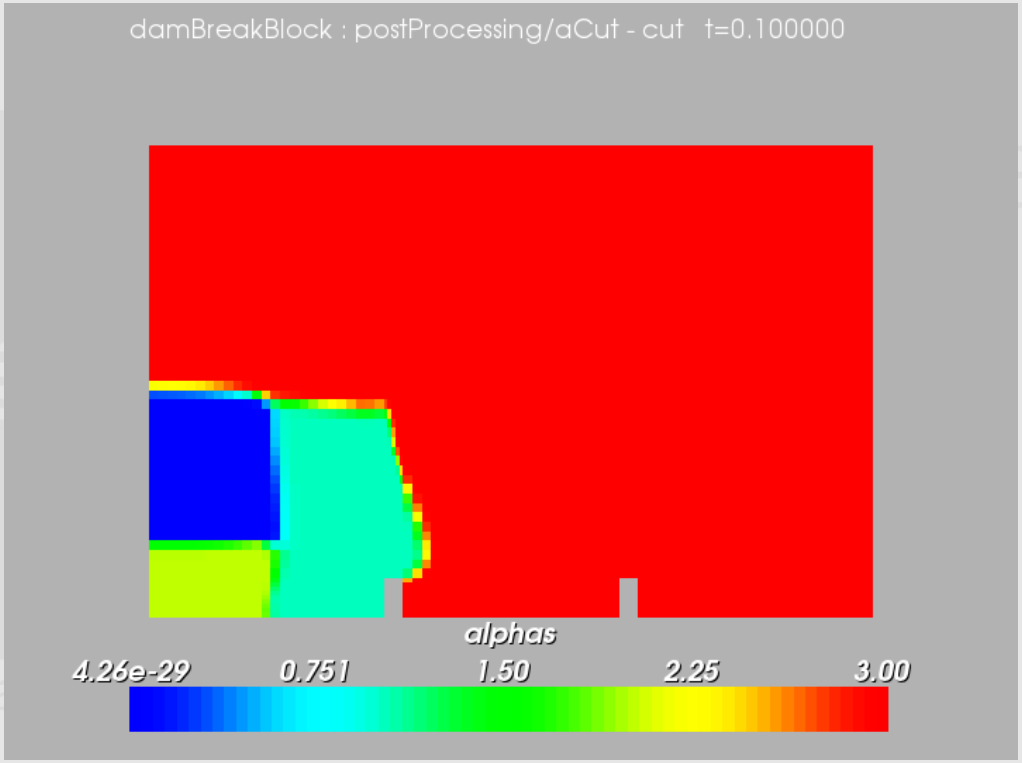


Figure : Honey moves slower

Phases at $t = 6$ - The tea variation

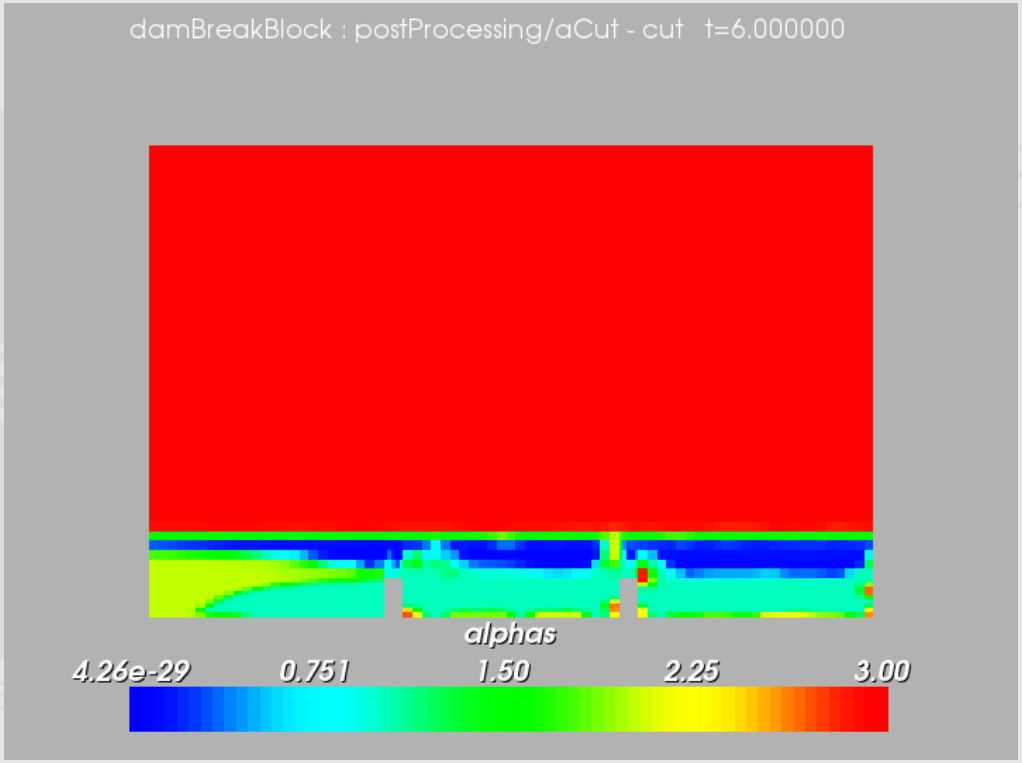


Figure : Honey on the bottom

Isidor pushes his work and Ignaz starts

```
> hg push
pushing to /Volumes/Foam/Cases/<brk>
    <cont>Scratch4KoreaPyFoam/VCS/damBreakBase
...
> cd ..
> pyFoamCloneCase.py damBreakBase <brk>
    <cont>damBreakEuler
updating to branch default
resolving manifests
getting .hgignore
...
getting system/setFieldsDict
22 files updated, 0 files merged, 0 files <brk>
    <cont>removed, 0 files unresolved
> cd damBreakEuler
> hg branch eulerSolver
```

Getting the files from the equivalent tutorial

```
> ECASE=$FOAM_TUTORIALS/multiphase/<brk>
    <cont>multiphaseEulerFoam/damBreak4phase
> cp $ECASE/0.org/* 0.org
> cp $ECASE/system/fvS* system
> cp $ECASE/constant/* constant
> hg status
M constant/transportProperties
M system/fvSchemes
M system/fvSolution
? 0.org/Uair
? 0.org/Umercury
? 0.org/Uoil
? 0.org/Uwater
? 0.org/p
? constant/LESProperties
? constant/MRFProperties
> hg add 0.org/* constant/*
```

Committing and running

```
> hg commit -m "The equivalen Euler-case"
0.org/Uair
0.org/Umercury
0.org/Uoil
0.org/Uwater
0.org/p
constant/LESProperties
constant/MRFProperties
constant/polyMesh/boundary
constant/transportProperties
system/fvSchemes
system/fvSolution
committed changeset 6:3067c96a66cc
> pyFoamPlotRunner.py --clear --progress <brk>
    <cont>multiphaseEulerFoam
```

Lower volume fractions

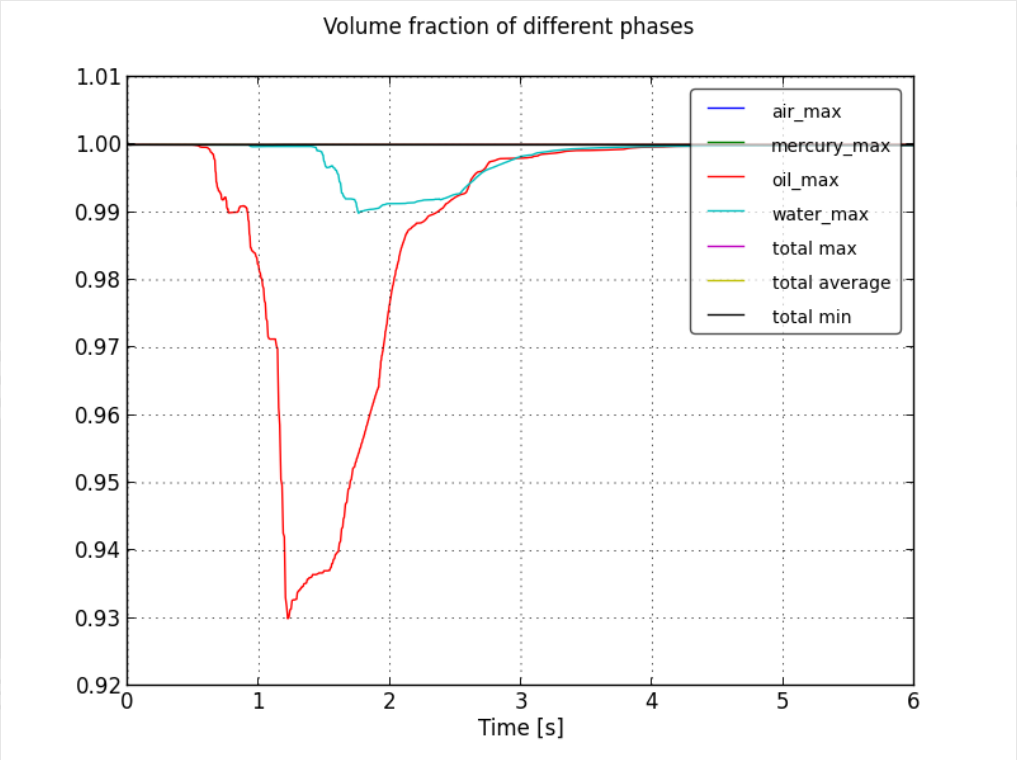


Figure : Maximum volume fractions are much lower than for VOF

Ignaz wants to get the geometry variation

... but not the materials (yet)

```
> hg tags
tip                                6:3067c96a66cc
meshTemplateFinished               3:6c6b0bc67056
> hg merge meshTemplateFinished
resolving manifests
removing constant/polyMesh/blockMeshDict
getting constant/polyMesh/blockMeshDict.template
merging constant/polyMesh/boundary
getting constant/polyMesh/meshStandardValues
2 files updated, 1 files merged, 1 files removed, 0 files <brk>
    <cont>unresolved
(branch merge, don't forget to commit)
> hg commit -m "Get the improved geometry"
constant/polyMesh/blockMeshDict.template
constant/polyMesh/boundary
constant/polyMesh/meshStandardValues
committed changeset 7:7432ad65a01d
```

The full tree

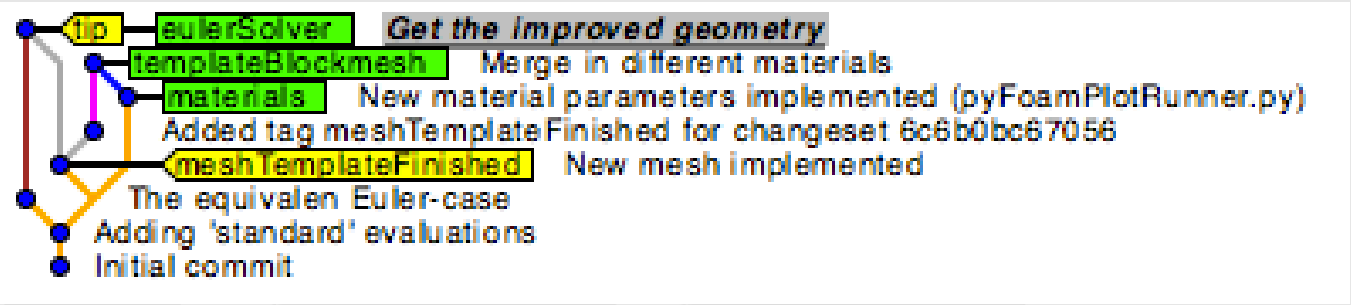


Figure : Visual tree of all commits in the repository

- A lot still has to be done for Ignaz, Isidor and Irma
- Version control helps them
  - to keep track of the changes they did
    - Even allows them to go back to versions that used to work and start a different branch there
  - Exchange improvements

Outline

Outline

The language

- Python is a scripting language
  - Object oriented
    - But also supports other paradigms like functional programming and aspect oriented programming
  - A big library that comes with it
  - Has a very simple syntax
- Built as the scripting-glue into a number of CAE-software
  - ParaView, VisIt, Salome, ...
- There is a number of interesting libraries for technical mathematical uses
  - `matplotlib`, `numpy`, `scipy`, `sympy`
  - A lot of them are glued together in the Mathematica-like program Sage

Three things to know about Python

If you've programmed in a procedural language before then reading `Python`-programs should be pretty straightforward except for

1. Indentation does the same thing as `{` and `}` for C++
2. `[]` usually is about lists (creation or element access)
3. `{}` creates a dictionary (lookup table with general keys)
   - whose elements can be accessed with `d[key]`
4. `self` is the same as `this` in C++ (the object itself)

PyFoam as a library

- Below the surface PyFoam is a library that knows how to write OpenFOAM-files
- The "workhorse" is the class `ParsedParameterFile`
  - Needs a filename as a parameter
    - Reads a (OpenFOAM)-dictionary-file
  - Can be used like a regular Python-dictionary
    - Accessing components with the `[]`-operator for reading and writing
    - Dictionaries and lists are mapped to the OpenFOAM dictionaries and lists
  - Manipulated dictionary can be written back with `writeFile()`

ParsedParameterFile

- This example
  - reads the old deltaT and prints it
  - resets deltaT to a thousandths of the endTime in the controlDict
  - writes the changed controlDict back to file

```
from PyFoam.RunDictionary.ParsedParameterFile <brk>
    <cont>import ParsedParameterFile

control=ParsedParameterFile("system/controlDict")
print "Timestep was",control["deltaT"]
controlDict["deltaT"]=controlDict["endTime"]/1000
print "Timestep now is",control["deltaT"]
control.writeFile()
```

Outline

What we didn't mention

- Passing pickled data between PyFoam-utilities via pipes
  - Only supported by certain applications
- `pyFoamPVSnapshot.py` for generating pictures with paraview without human interaction
  - A Paraview state file (PVSM) is required
- The built-in network server
  - Well. We had a quick glance ...
- Manipulating files with utilities
- other stuff I forgot

Outline

Most important things to remember about PyFoam

1. All utility names start with `pyFoam`
2. Documentation is available with the option `--help`
3. An overview is found on `openfoamwiki.net`
4. Help is available on the Message-Board
   - **Hint:** The word PyFoam in the title fill increase the chances of it being seen
5. Releases are announced also on the MessageBoard
   - And on the Twitter-channel `@swakPyFoam`

Goodbye to Ignaz

- Ignaz work with the new project is far from complete
  - The results are not very impressive
  - There are still numerical problems
- Maybe we'll hear from him next year
  - And maybe by then he can pour tea correctly

Goodbye to you

Thanks for listening
Questions?

License of this presentation

Authors of this document are:

Bernhard F.W. Gschaider original author and responsible for the strange English grammar. Contact him for a copy of the sources if you want to extend/improve/use this presentation