Automatization with pyFoam

How Python helps us to avoid contact with OpenFOAM™

Bernhard F.W. Gschaider
bgschaid@ice-sf.at

ICE Strömungsforschung

Innovative Computational Engineering

5th OpenFOAM™-Workshop, Gothenburg
24.June 2010

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Overview

Aim of this presentation

This presentation shows how to use PyFoam to run a case automatically and evaluate the results

It does not give

- a full overview of Python
- an overview of all the capabilities of PyFoam

It assumes

- that you know your way around OpenFOAM™
- have programmed a little bit in **some** language (not necessarily Python)

What is PyFoam

PyFoam is

1. a Python library that
   - manipulates OpenFOAM™-cases
   - controls OpenFOAM™-runs
2. Utilities based on that library

What happened before

- Last year in Montreal a presentation PyFoam - Happy foaming with Python (URL below) was given
  - It is not necessary that you have read it
  - .... but helpful (Now it's to late. Read it later.)
- We were introduced to the CFD-engineer Ignaz Gartengschirrl
  - A specialist on calculating the `damBreak`-case
  - ... using PyFoam
- During the last year Ignaz extended
  - his professional range
  - his knowledge of PyFoam

`http://www.openfoamworkshop.org/2009/4th_Workshop/0_Feature_Presentations/OFW4_2009_Gschaider_PyFoam.pdf`

Conventions

Whenever Ignaz (and also you) writes something

- on the shell
- in an editor

it will be found in a coloured box. Sometimes there will be output, too

Ignaz writes on the shell

```
> date
Fri May 15 01:56:12 CEST 2009
```

Ignaz's Python-code

```
sum=0
for v in [7,13,42]:
    sum+=v
print "The sum is",sum
# this is a long line that will be <brk>
    <cont>continued in the next line
```

Ignaz edits a file

```
fooCoeffs {
  bar 23;
  lst ( inlet outlet );
}
```

Getting the git-archive with the data

- The data and programs needed are in a `tar`-file on the stick
  - If you haven't booted from the stick the paths may be different for you
- First we copy it to your home and create a working directory:

Preparing for work

```
> cd $HOME
> mkdir pyFoamAdvanced
> cd pyFoamAdvanced
> tar xzf /cdrom/OFW5/Advanced_Training/pyFoamAdvanced.git.tgz
```

- Now there should be a directory `pyFoamAdvanced.git`

Getting to a specific step

- What we extracted is a GIT-archive
- There are several branches for different phases
  - So if you couldn't follow you can catch up

Going to a specific step

```
> cd $HOME/pyFoamAdvanced
> git clone pyFoamAdvanced.git -b step1manualStarting step1
```

- Now there should be a directory `step1`

What is Python

PYTHON

- Is a scripting language
  - No compilation required
- Is object-oriented
- Comes *batteries included*: has a large standard-library for many common tasks
  - Non essential parts (like regular expressions) were moved to the library
- Widely used
  - Pre-installed on most LINUX-systems because many system tools (installers for instance) use it
  - Becomes *scripting language of choice* for a number of programs (amongst others the post-processors PARAVIEW and VISIT and the pre-processor SALOME)

3 things you have to know about Python

... to understand the programming examples

1. Indentation does the same thing `{` and `}` do for C++
2. `[]` signifies a list (which is an array)
3. `{}` is a dictionary (whose elements are accessed with `[key]`)
4. `self` is the same as `this` in C++ (The object itself)

Aeh. The 4 things to know about PYTHON *are ....*

The Python Shell

- PYTHON-files usually end with the extension `.py`

Running a Python-file

```
> python test.py
```

- But you can also use it interactively

Using the Python-shell

```
> python
Python 2.6.5 (r265:79063, May 18 2010, 10:54:45)
[GCC 4.2.1 (Apple Inc. build 5659)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>> import sys
>>> print sys.platform
darwin
>>>
```

exit with `Ctrl+D`

Basic constructs - lists and dictionaries

Basic usage of lists and dictionaries

```
>>> lst=[2,"interFoam",1.3]
>>> lst[0]
2
>>> lst+=["nothing"]
>>> print lst
[2, 'interFoam', 1.3, 'nothing']
>>> order={'interFoam':1,'blockMesh':0,'paraFoam':2}
>>> order['interFoam']
1
>>> 'paraFoam' in order
True
>>> order['setFields']=0.5
>>> print order
{'blockMesh': 0, 'interFoam': 1, 'setFields': 0.5, 'paraFoam': 2}
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Basic constructs - loops

Two kinds of loops

```
>>> vals=[2,3,5,7,11,13,17,19]
>>> for v in vals:
...     print v*v,
...
4 9 25 49 121 169 289 361
>>> i=0
>>> while i<len(vals):
...     vals[i]="no"
...     i+=2
...
>>> print vals
['no', 3, 'no', 7, 'no', 13, 'no', 19]
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Basic constructs - branches

One kind of branch is sufficient

```
>>> for i in range(-3,3):
...     if i==0:
...         print "zero",
...     elif (i%2)==0:
...         print "even",
...     else:
...         print "odd",
...
odd even odd zero odd even
```

Basic constructs - classes

Ignaz tells us all he knows

```
>>> class Ignaz:
...     def __init__(self,val):
...         self.val = val
...     def __str__(self):
...         return "Ignaz says "+str(self.val)
...     def forget(self):
...         self.val = None
...
>>> ig=Ignaz(1+1)
>>> print ig
Ignaz says 2
>>> ig.forget()
>>> print ig
Ignaz says None
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Basic constructs - importing libraries

Standing on the shoulders of ... other programmers

```
>>> from time import time
>>> time()
1275658314.7988529
>>> import sys
>>> dir(sys)
['__displayhook__', '__doc__', '__egginsert', '__excepthook__', '__name__', '<brk>
    <cont>__package__', '__plen', '__stderr__', '__stdin__', '__stdout__', '<brk>
    <cont>_clear_type_cache', '_current_frames', '_getframe', 'api_version', '<brk>
    <cont>argv', 'builtin_module_names', 'byteorder', 'call_tracing', '<brk>
    <cont>callstats', 'copyright', 'displayhook', 'dont_write_bytecode', '<brk>
    <cont>exc_clear', 'exc_info', 'exc_type', 'excepthook', 'exec_prefix', '<brk>
    <cont>executable', 'exit', 'flags', 'float_info', 'getcheckinterval', '<brk>
    <cont>getdefaultencoding', 'getdlopenflags', 'getfilesystemencoding', '<brk>
    <cont>getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof', '<brk>
    <cont>gettrace', 'hexversion', 'last_type', 'last_value', 'maxint', '<brk>
    <cont>maxsize', 'maxunicode', 'meta_path', 'modules', 'path', 'path_hooks',<brk>
    <cont> 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2', '<brk>
    <cont>py3kwarning', 'setappdefaultencoding', 'setcheckinterval', '<brk>
    <cont>setdlopenflags', 'setprofile', 'setrecursionlimit', 'settrace', '<brk>
    <cont>stderr', 'stdin', 'stdout', 'subversion', 'version', 'version_info', <brk>
    <cont>'warnoptions']
>>> print sys.version
2.6.5 (r265:79063, May 18 2010, 10:54:45)
[GCC 4.2.1 (Apple Inc. build 5659)]
```

How to get help - On the shell

What is that ParsedParameterFile anyway

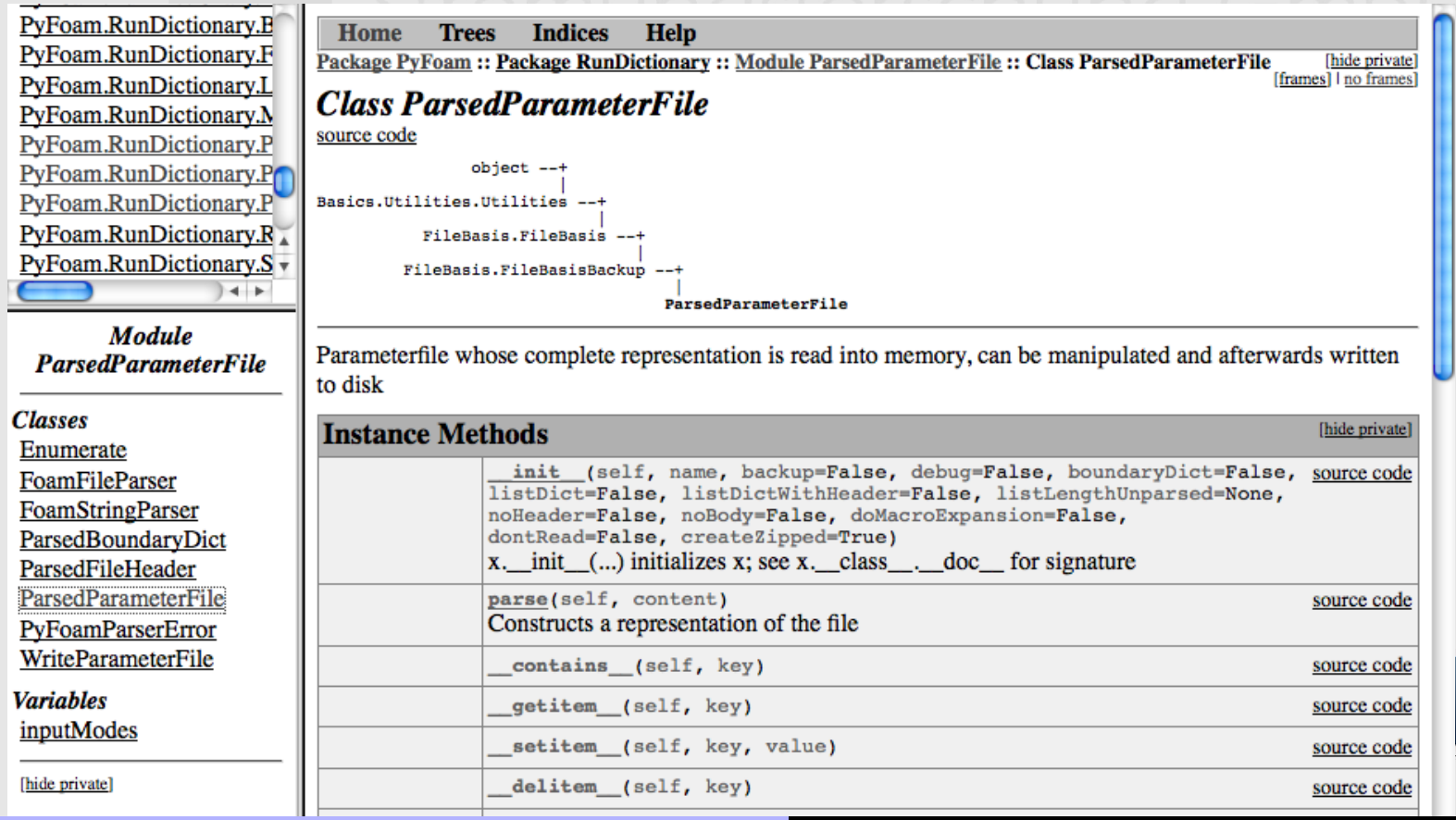
```
>>> from PyFoam.RunDictionary.ParsedParameterFile import <brk>
    <cont>ParsedParameterFile
>>> print ParsedParameterFile.getCaseDir.__doc__
Return the path to the case of this file (if any valid case is found).
        Else return None
>>> help(ParsedParameterFile
Help on class ParsedParameterFile in module PyFoam.RunDictionary.<brk>
    <cont>ParsedParameterFile:

class ParsedParameterFile(PyFoam.RunDictionary.FileBasis.FileBasisBackup)
 |  Parameterfile whose complete representation is read into
 |  memory, can be manipulated and afterwards written to disk
 |
 |  Method resolution order:
...
```

- All these texts were extracted from the source:
  - PYTHON comes with an included DOXYGEN

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Help on PyFoam

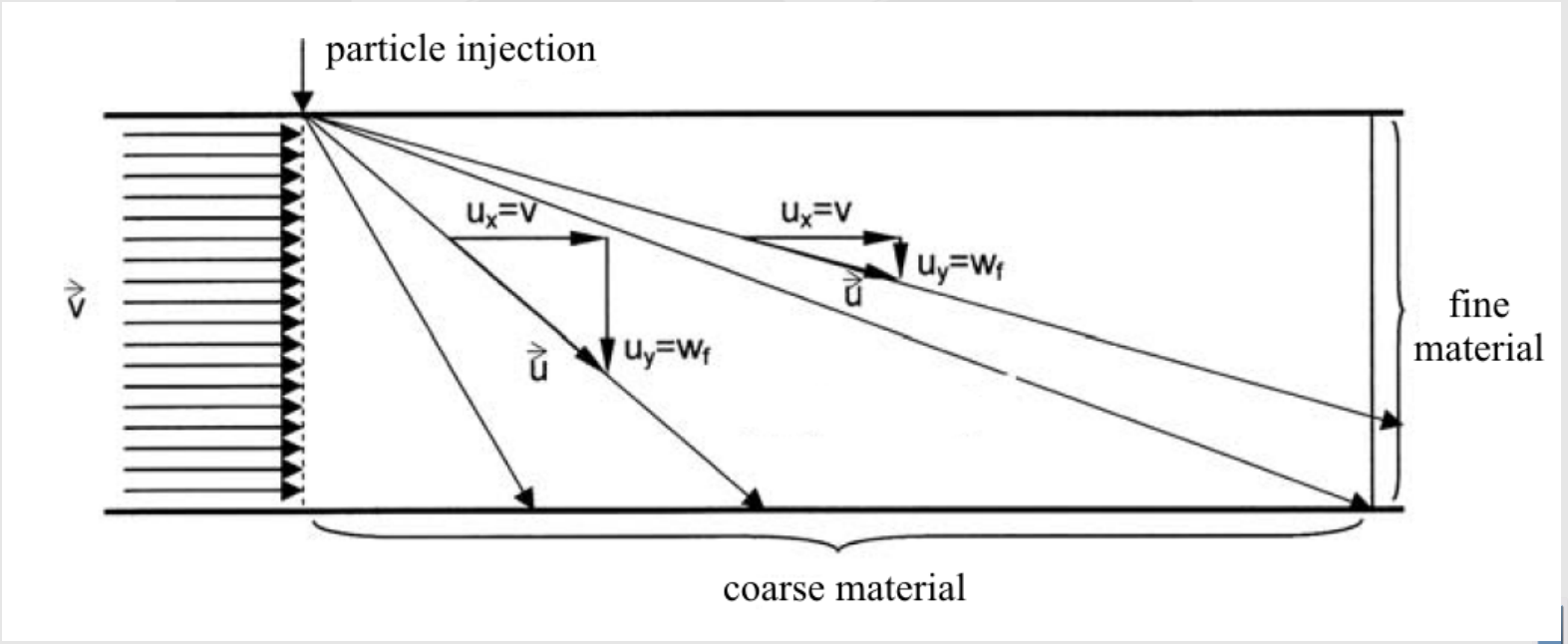
- With the help of a program called epydoc a more user friendly HTML-documentation can be produced
  - This documentation is included in the normal distribution



Strömungsforschung GmbH
CE
ative Computational Engineering

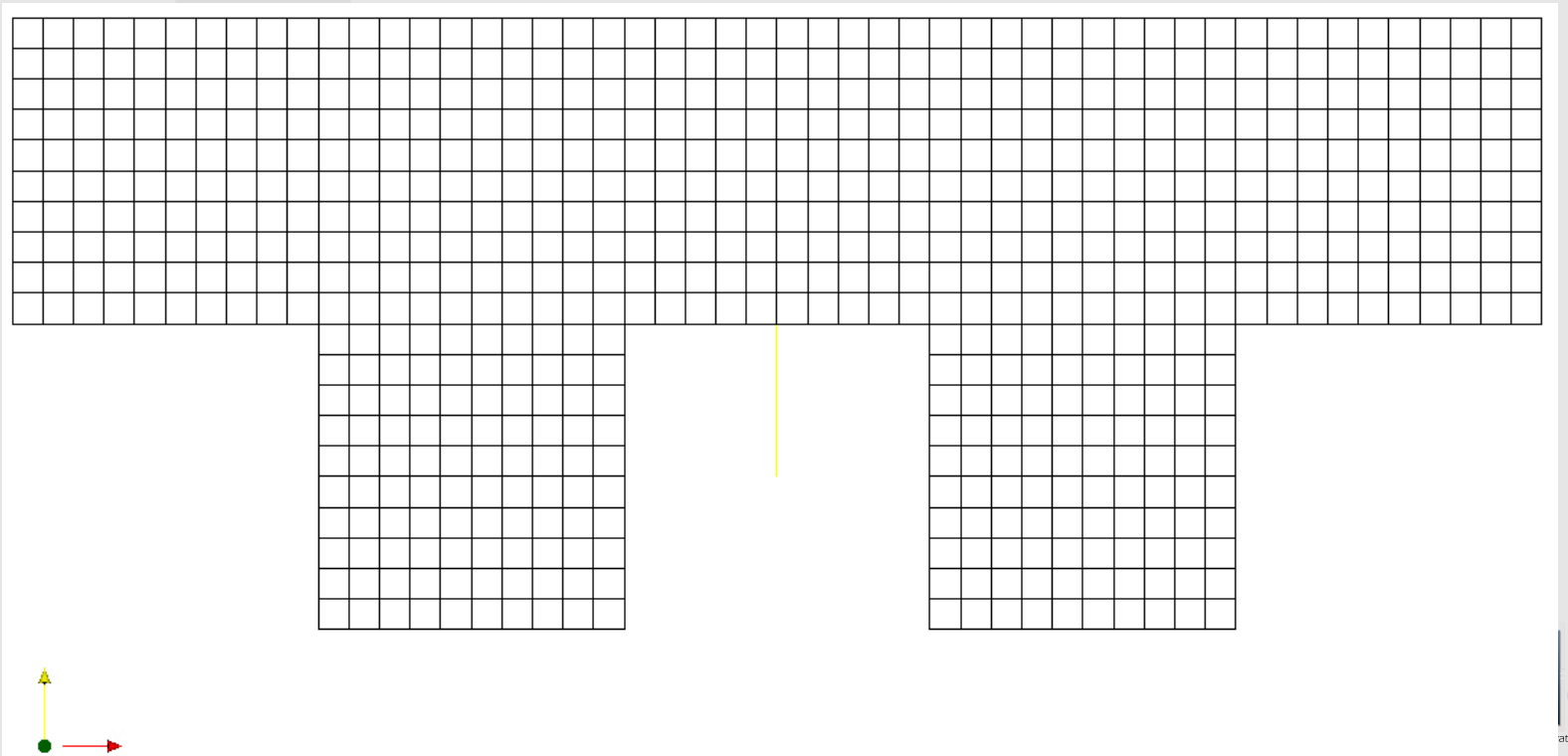
Separating particles of different size

- Ignaz is asked to design a particle separator
- He knows the physics: Drag-force vs. gravity



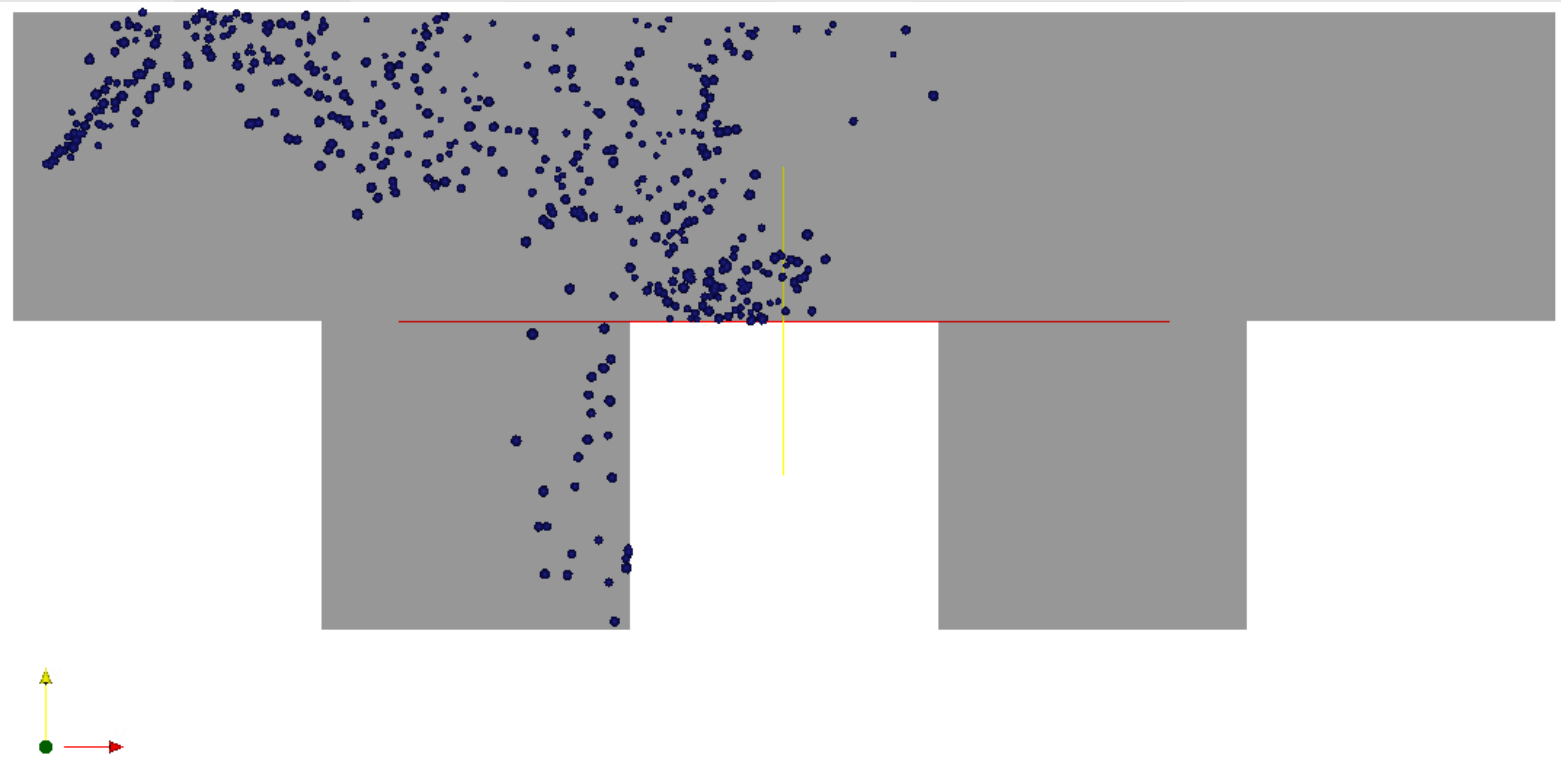
The simplified particle separator

- After 2 weeks the CAD-engineer came up with this complex design
  - Inlet and particle injector at the left



What a solution looks like

- The goal is: Change the inlet velocity of the channel in such a way that
  - The number of particles in the second bin maximizes
  - No (or as few as possible) particles leave through the oultet



Basis: rhoPisoTwinParcelFoam

- Our solver is based on `rhoPisoTwinParcelFoam` which can be found in the tutorials. It is
  - transient
  - compressible
  - turbulent
  - has two different populations of particles
- Changes in `rhoPisoAbscheidParcelFoam` are
  - Only one particle population
  - New particle class

Where do the particles go

- The whole solver had to be written for this method:

In basicAbscheideParcel.C

```
1  void Foam::basicAbscheideParcel::hitPatch
   (
3      const polyPatch& p,trackData& td
   )
5  {
       Pout << "Abgeschieden at " << p.name() << " with m: " << mass()
7           << " N: " << nParticle() << " = total: "
           << mass()*nParticle() <<  endl;
9      KinematicParcel<basicAbscheideParcel>::hitPatch(p,td);
   }
```

- When a particle hits a patch (not a wall), this is the output:

Output on the terminal

```
Abgeschieden at outlet with m: 3.605329877e-10 N: 41386.01459 = total: 1.492102349e-05
```

- We notice that Ignaz never bothered to look up the English word for abscheiden in his dictionary

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Building the solver

1. Get the first stage (if we haven't already done so)

Getting it with Git

```
> cd $HOME/pyFoamAdvanced
> git clone pyFoamAdvanced.git -b step1manualStarting step1
> cd step1
```

2. Compiling the solver (we only have to do this once)

Making sure we use the right OpenFOAM™-version

```
> pyFoamExecute.py --foam=1.5-dev wmake solver/rhoPisoAbscheidParcelFoam15
 PyFoam WARNING on line 152 of file /software/PyFoam/FoamInformation.py : 1.5-dev is <brk>
      <cont>already being used
```

Compiling it for debugging

```
> pyFoamExecute.py --foam=1.5-dev --force-debug wmake solver/rhoPisoAbscheidParcelFoam15
```

A clean slate

- Ignaz likes to start with a new case

Making a copy

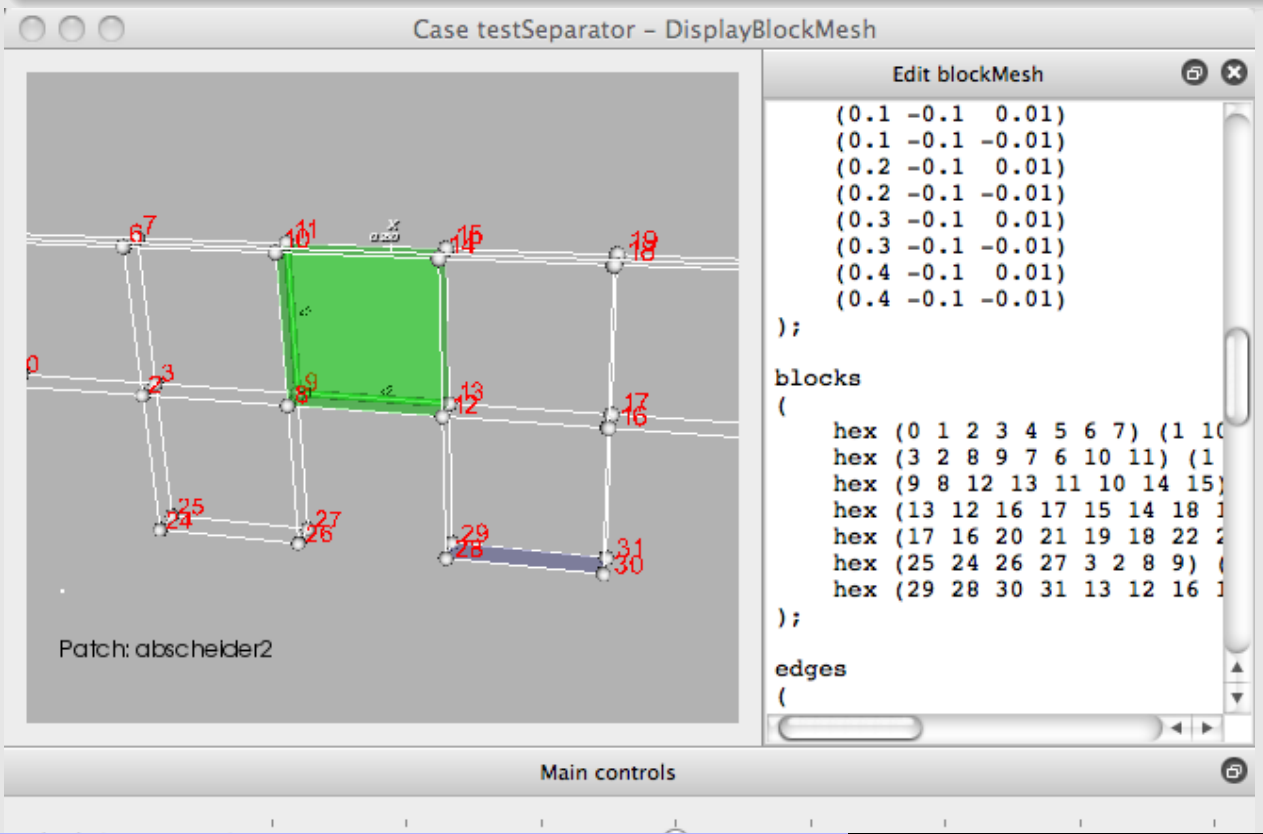
```
> pyFoamCloneCase.py --add=default.pvsm cases/twoCavityChannel <brk>
    <cont>testSeparator
PyFoam WARNING on line 85 of file /Users/bgschaid/private_python/PyFoam/<brk>
    <cont>Applications/CloneCase.py : Directory  does not exist. Creating
> cd testSeparator
```

- Now he can modify the case any way he likes

Editing the blockMesh

Making `blockMesh` less painful

```
1 > pyFoamDisplayBlockMesh.py constant/polyMesh/blockMeshDict
```



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Checking the boundary conditions

All boundary conditions at once

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

Table of boundary conditions for t = 0

          | abscheider1  | abscheider2  | bottom       | defaultFaces | inlet        | outlet       | top
-------------------------------------------------------------------------------------------------------------
Patch Type | patch        | patch        | wall         | empty        | patch        | patch        | wall
Length     | 10           | 10           | 70           | 1400         | 10           | 10           | 50
-------------------------------------------------------------------------------------------------------------
T          | inletOutlet  | inletOutlet  | zeroGradient | empty        | fixedValue   | zeroGradient | zeroGradient
U          | fixedValue   | fixedValue   | fixedValue   | empty        | fixedValue   | inletOutlet  | fixedValue
epsilon    | inletOutlet  | inletOutlet  | zeroGradient | empty        | fixedValue   | inletOutlet  | zeroGradient
k          | inletOutlet  | inletOutlet  | zeroGradient | empty        | fixedValue   | inletOutlet  | zeroGradient
p          | zeroGradient | zeroGradient | zeroGradient | empty        | zeroGradient | fixedValue   | zeroGradient
```

Other things the tool can report

```
> pyFoamCaseReport.py --help
```

Preparing for parallel runs

2 CPUs is all we have

```
> pyFoamDecompose.py . 2
...
> pyFoamCaseReport.py --decomposition .
Case is decomposed for 2 processors
              |      0      1
------------------------------
      Points  |    814    770
       Faces  |   1486   1404
       Cells  |    360    340
------------------------------
 abscheider1  |     10      0
 abscheider2  |      0     10
      bottom  |     36     34
defaultFaces  |    720    680
       inlet  |     10      0
      outlet  |      0     10
         top  |     26     24
```

Hey! Ho! Let's go!

At last Ignaz is eager to see some numbers ...

Removing old data and starting the run

```
> pyFoamRunner.py --progress --proc=2 --clear rhoPisoAbscheidParcelFoam
 Reading regular expressions from /Volumes/Rest/StickTest/Shared/AdvanzedWork/step1/<brk>
      <cont>testSeparator/customRegexp
Clearing out old timesteps ....
 t =  0.0736599

...

> less PyFoamRunner.rhoPisoAbscheidParcelFoam.logfile
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
Exec   : rhoPisoAbscheidParcelFoam
...
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Regular expressions in 3 minutes

- Regular expressions are essential for `customRegexp` (therefor the name)
- Usually a letter is matched by itself
  - `.` matches any letter
- Patterns are enclosed by `(` and `)`
  - Patterns are numbered by the order in which they appear in the expression
- `+` means "repeat the previous pattern at least once"
- `%f%` is PyFoam-shorthand for a regular expression that matches any floating-point number

Parcels in the system

Ignaz wants a plot of the parcels in the system

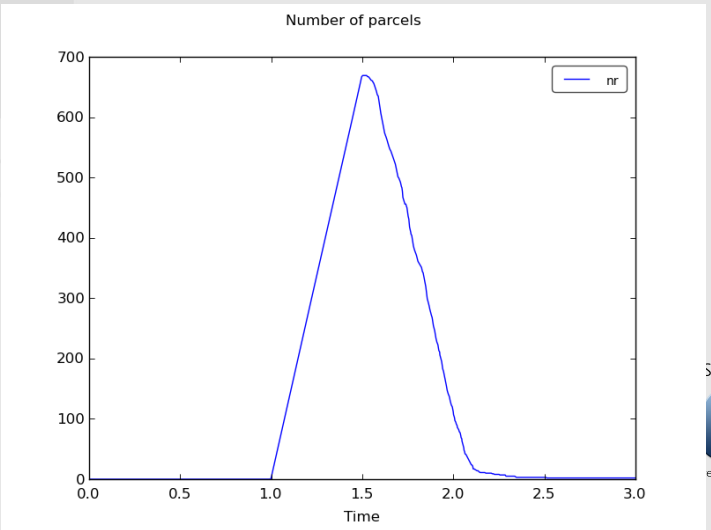
expr The regular expression that should be matched

theTitle Title of the plot

name List of annotations

Simple plot

```
parcels
{
        theTitle "Number of parcels";
        expr "Current number of <brk>
             <cont>parcels        =  (%f <brk>
             <cont>%)";
        titles ( nr );
}
```



Mass leaving the system

List of exits is not previously known. `type dynamic` allows to dynamically build it

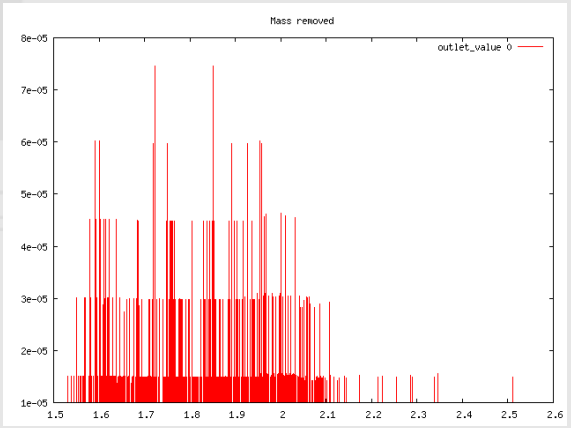
idNr The pattern number of the ID (name)

accumulation `sum` means that several occurences of a pattern in a time-step will be added (usually only the first is used)

with How the data should be plotted (see `with` in GnuPlot)

Simple plot

```
amount
{
        theTitle "Mass removed";
        expr "Abgeschieden at (.+) <brk>
              <cont>with .+total: (%f%) <brk>
              <cont>";
        type dynamic;
        idNr 1;
        with impulses;
        accumulation sum;
}
```



Using it to plot

Displaying from a logfile

```
1 > pyFoamPlotWatcher.py PyFoamRunner.rhoPisoAbscheidParcelFoam.logfile
```

Saving pictures afterwards

```
1 > pyFoamPlotWatcher.py --progress PyFoamRunner.rhoPisoAbscheidParcelFoam.logfile --<brk>
      <cont>hardcopy
```

Using matplotlib

```
1 > pyFoamPlotWatcher.py --progress PyFoamRunner.rhoPisoAbscheidParcelFoam.logfile --<brk>
      <cont>hardcopy --implementation=matplotlib
```

Or using stuff stored by the pyFoamPlotRunner

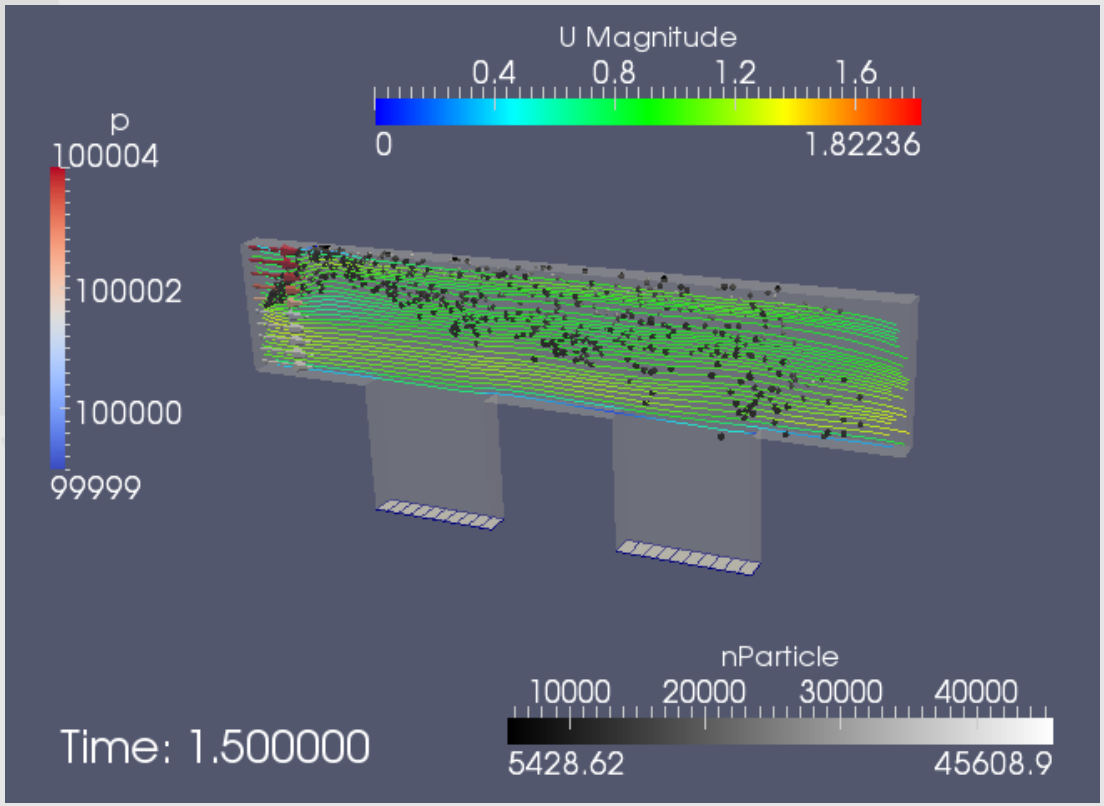
```
1 > pyFoamRedoPlot.py --pickle PyFoamRunner.rhoPisoAbscheidParcelFoam.analyzed/<brk>
      <cont>pickledPlots
```

mungsforschung GmbH
putational Engineering

Ignaz prepared a complex visualization

He saved a ParaView-statefile named `default.pvsm`

```
1 > pyFoamPVSnapshot.py --time=1.5 .
```



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Other things to do with state-files

If you haven't cloned `default.pvsm`

```
> pyFoamPVSnapshot.py --time=1.5 . --state=../cases/twoCavityChannel/<brk>
    <cont>default.pvsm
```

Plotting more than one time ... for a poster

```
> pyFoamPVSnapshot.py --time=2 --time=1.5 . --maginfication=3
```

Starting ParaView with that state

```
> pyFoamPVLoadState.py . --state=../cases/twoCavityChannel/default.pvsm
```

ng GmbH

Innovative Computational Engineering

Exercise - What did I do?

- Have a look at the `PyFoamHistory`-file in the case
- What does it tell you?
- Who has the fewest lines?
- Who has the most?

Preparing for a new take

1. Get the second stage

Getting it with Git

```
> cd $HOME/pyFoamAdvanced
> git clone pyFoamAdvanced.git -b step2template step2
> cd step2
```

2. A clean copy

Cloning again

```
> pyFoamCloneCase.py --add=default.pvsm cases/twoCavityChannel <brk>
    <cont>testSeparator
 PyFoam WARNING on line 85 of file /Users/bgschaid/private_python/PyFoam/<brk>
      <cont>Applications/CloneCase.py : Directory  does not exist. Creating
> cd testSeparator
```

Getting to the heart of PyFoam

- Now Ignaz wants to prepare his case for automatic setup
  - Changing the inlet velocity
  - Changing the geometry
- Before testing a script we try it with commands

Changing the velocity

This is the part of `0/U` that Ignaz does not want to touch with his text editor

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }
...
```

Reading a dictionary

This abstract example shows how a dictionary-file is "seen" inside a PYTHON-program

The dictionary testDict

```
model theModel;

patches ( inlet
          outlet );

theModelCoeffs {
        alpha 0.3;
        beta 2 inlet ;
}
```

The Python-code

```
from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile

file=ParsedParameterFile("testDict")

# none of these should fail
assert file["model"]=="theModel"
assert len(file["patches"])==2
assert file["patches"][0]=="inlet"
assert file["theModelCoeffs"]["beta"][1]=="inlet"

# manipulation is possible
file["theModelCoeffs"]["alpha"]+=1
file["newCoeffs"]={"a":1,"b":[2,3,4]}

print file
```

Trying half the original velocity

Not the most elegant way to set the velocity

```
> pyFoamWriteDictionary.py 0/U "boundaryField['inlet']['value']" "uniform <brk>
    <cont>(0.5 0 0)" --strip-quotes-from-value
```

Ignaz checks what has changed

```
1  > pyFoamCompareDictionary.py 0/U ../cases/twoCavityChannel
   PyFoam WARNING on line 90 of file /software/PyFoam/Applications/<brk>
        <cont>CompareDictionary.py : Found /data/step2/cases/twoCavityChannel<brk>
        <cont>/0/U and using this
3  >>>< Field U[boundaryField][inlet][value] : Differs
   >>Source:
5  uniform (0.5 0 0)
   <<Destination:
7  uniform (1 0 0)
```

How templates work

- The template file is called the same as the file it will produce
  - Just with `.template` appended
- Lines that start with $$ are variable declarations
  - They don't appear in the final output
- Strings between $ and $ are treated as PYTHON-expressions and evaluated
  - Previously declared variables are used
- Everything else is copied to the output

Declaring helper variables

In constant/polyMeshDict.template

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * <brk>
     <cont>* //

$$ x1=preLen
$$ x2=preLen+cavityWidth
$$ x3=preLen+cavityWidth+interLen
$$ x4=preLen+2*cavityWidth+interLen
$$ x5=2*preLen+2*cavityWidth+interLen
$$ y1=channelHeight
$$ y2=-cavityDepth

convertToMeters 1.0;

...
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The actual points

Later in constant/polyMeshDict.template

```
...
vertices
(
    (0     0    -0.01)
    (0     0     0.01)
    ($x1$  0     0.01)
    ($x1$  0    -0.01)
    (0     $y1$ -0.01)
    (0     $y1$ 0.01)
    ($x1$  $y1$ 0.01)
    ($x1$  $y1$ -0.01)
    ($x2$   0     0.01)
    ($x2$   0    -0.01)
...
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The blocks

Further down in constant/polyMeshDict.template

```
....
blocks
(
    hex (0 1 2 3 4 5 6 7) (1 $int(ceil(preLen/dx))$ $int(ceil(channelHeight<brk>
        <cont>/dx))$) simpleGrading (1 1 1)
    hex (3 2 8 9 7 6 10 11) (1 $int(ceil(cavityWidth/dx))$ $int(ceil(<brk>
        <cont>channelHeight/dx))$) simpleGrading (1 1 1)
    hex (9 8 12 13 11 10 14 15) (1 $int(ceil(interLen/dx))$ $int(ceil(<brk>
        <cont>channelHeight/dx))$) simpleGrading (1 1 1)
    hex (13 12 16 17 15 14 18 19) (1 $int(ceil(cavityWidth/dx))$ $int(ceil(<brk>
        <cont>channelHeight/dx))$) simpleGrading (1 1 1)
    hex (17 16 20 21 19 18 22 23) (1 $int(ceil(preLen/dx))$ $int(ceil(<brk>
        <cont>channelHeight/dx))$) simpleGrading (1 1 1)
    hex (25 24 26 27 3 2 8 9) (1 $int(ceil(cavityWidth/dx))$ $int(ceil(<brk>
        <cont>cavityDepth/dx))$) simpleGrading (1 1 1)
    hex (29 28 30 31 13 12 16 17) (1 $int(ceil(cavityWidth/dx))$ $int(ceil(<brk>
        <cont>cavityDepth/dx))$) simpleGrading (1 1 1)
);
....
```

Converting the template

The secret is to get a complete Python-dictionary into a string

```
> pyFoamFromTemplate.py constant/polyMesh/blockMeshDict "{'preLen':0.2,'<brk>
    <cont>cavityWidth':0.15,'channelHeight':0.1,'interLen':0.05,'<brk>
    <cont>cavityDepth':0.02,'dx':0.005}"
```

Produces a constant/polyMesh/blockMeshDict with lines like

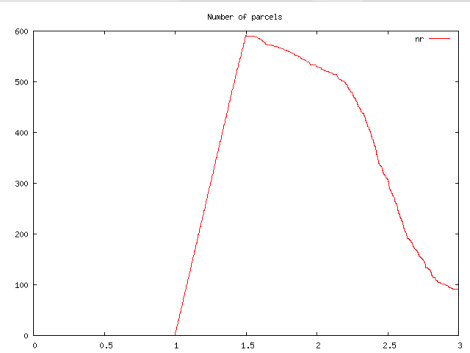
```
blocks
(
    hex (0 1 2 3 4 5 6 7) (1 40 20) simpleGrading (1 1 1)
    hex (3 2 8 9 7 6 10 11) (1 30 20) simpleGrading (1 1 1)
    hex (9 8 12 13 11 10 14 15) (1 10 20) simpleGrading (1 1 1)
    hex (13 12 16 17 15 14 18 19) (1 30 20) simpleGrading (1 1 1)
    hex (17 16 20 21 19 18 22 23) (1 40 20) simpleGrading (1 1 1)
    hex (25 24 26 27 3 2 8 9) (1 30 4) simpleGrading (1 1 1)
    hex (29 28 30 31 13 12 16 17) (1 30 4) simpleGrading (1 1 1)
);
```

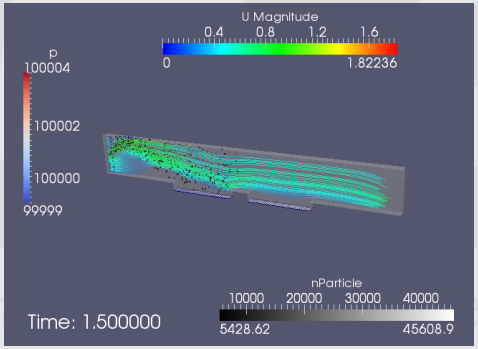
ng GmbH

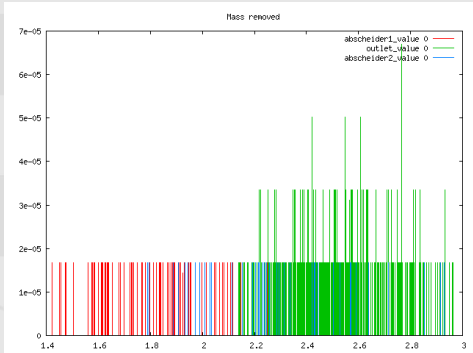
Isidor wants to see the results

Running it

```
> blockMesh
> pyFoamPlotRunner.py --hardcopy --progress --clear <brk>
    <cont>rhoPisoAbscheidParcelFoam
> pyFoamPVSnapshot.py --time=1.5 .
```







Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Cloning again

1. Get the third stage

Getting it with `Git`

```
> cd $HOME/pyFoamAdvanced
> git clone pyFoamAdvanced.git -b step3shellScript step3
> cd step3
```

2. This time no `pyFoamCloneCase.py`

One script to bind them all

- Ignaz knows the commands by heart
- .. but doesn't want to execute them every time by hand
- So he copies them into a script
- .. and adds some *shell sugar*

The shell script

The file scripts/prepareRun.sh

```
1  #! /bin/sh
3  TEMPLATE=$1
   TARGET=$2
5  VELOCITY=$3
   DX=$4
7
   GEOMETRY="{'preLen':0.2,'cavityWidth':0.15,'channelHeight':0.1,'interLen':0.05,'cavityDepth<brk>
        <cont>':0.02,'dx':$DX}"
9
   echo "Creating case $TARGET"
11
   pyFoamCloneCase.py $TEMPLATE $TARGET
13 pyFoamFromTemplate.py $TARGET/constant/polyMesh/blockMeshDict $GEOMETRY
   pyFoamWriteDictionary.py $TARGET/0/U "boundaryField['inlet']['value']" "uniform ($VELOCITY 0 0)"<brk>
        <cont> --strip-quotes-from-value
15
   echo "Calling blockMesh and decompose"
17
   pyFoamRunner.py --clear --progress blockMesh -case $TARGET
19 pyFoamDecompose.py --progress $TARGET 2
21 pyFoamCaseReport.py --decomposition $TARGET
23 echo "Running"
25 pyFoamRunner.py --progress --proc=2  --foam=1.5-dev rhoPisoAbscheidParcelFoam -case $TARGET
```

Executing it

... is easy

```
> ./scripts/prepareRun.sh cases/twoCaviyChannel testSeparator 0.7 0.01
Creating case testSeparator
...
```

That's all you hear about shell-scripts here

There's more to `git` than clone

... but we don't use it

The only `git` command you see in this presentation

```
> cd $HOME/pyFoamAdvanced
> git clone pyFoamAdvanced.git -b step4pythonScript step4
> cd step4
```

Script reloaded

- Now Ignaz becomes ambitious
- He wants to do it better ... in pure PYTHON
- The end result is longer
- ... but will be much easier to modify

Reading the parameters

No PyFoam here - Start of scripts/prepareRun.py

```
1  #! /usr/bin/env python

3  import sys
   from os import path
5
   if len(sys.argv)<4:
7      print "Need␣<template>␣<target>␣<velocity>␣[<dx>]"
       sys.exit(-1)
9
   template=sys.argv[1]
11 target=sys.argv[2]
   velocity=float(sys.argv[3])
13
   dx=0.01
15 if len(sys.argv)>4:
       dx=float(sys.argv[4])
```

- sys.argv is the list of command-line parameters

Cloning the directory

First time we use the library - scripts/prepareRun.py

```
from PyFoam.RunDictionary.SolutionDirectory import SolutionDirectory

orig=SolutionDirectory(template,
                       archive=None,
                       paraviewLink=False)
work=orig.cloneCase(target)
```

- SolutionDirectory represents an OpenFOAM™-case
  - Including time-steps etc
- cloneCase returns another SolutionDirectory

Using the template

The dictionary now looks more natural - scripts/prepareRun.py

```
from PyFoam.Basics.TemplateFile import TemplateFile

bmName=path.join(work.polyMeshDir(),"blockMeshDict")

template=TemplateFile(bmName+".template")
template.writeToFile(bmName,{'preLen':0.2,
                             'cavityWidth':0.15,
                             'channelHeight':0.1,
                             'interLen':0.05,
                             'cavityDepth':0.02,
                             'dx':dx})
```

- os.path is a convenient way to keep pathnames portable
  - This line should need no change on Windows (never tested it though)

Modifying the velocity

Dictionary modification is trivial - scripts/prepareRun.py

```
1  from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile
   from PyFoam.Basics.DataStructures import Vector
3
   velFile=ParsedParameterFile(path.join(work.initialDir(),"U"))
5  velFile["boundaryField"]["inlet"]["value"].setUniform(Vector(velocity,0,0))
   velFile.writeFile()
```

- ParsedParameterFile is the workhorse of PyFoam
- Mastering it is easy:
  1. Create it. The file is read
  2. Manipulate it like a regular Python-dictionary
  3. Write it back

Administrative stuff

Make sure we use the right version - scripts/prepareRun.py

```
from PyFoam.FoamInformation import changeFoamVersion
changeFoamVersion("1.5-dev")

from PyFoam.Error import error
```

- changeFoamVersion makes sure that the *right* OpenFOAM™-version is used
- error prints an error message and aborts the program

Running `blockMesh`

Running and checking - `scripts/prepareRun.py`

```
from PyFoam.Execution.BasicRunner import BasicRunner
blockRun=BasicRunner(argv=["blockMesh",
                           "-case",work.name],
                     silent=True,
                     server=False,
                     logname="Blocky")
print "Running blockMesh"
blockRun.start()
if not blockRun.runOK():
    error("There was a problem with blockMesh")
```

- BasicRunner runs any OpenFOAM™-command
  - **argv** Command-line passed to OpenFOAM™
  - **silent** No output to the terminal
  - **server** No server-process that controls the execution
  - **logname** Name of the logfile

Decomposing

Using the functionality of a command line-tool - scripts/prepareRun.py

```
from PyFoam.Applications.Decomposer import Decomposer
print "Decomposing"
Decomposer(args=["--progress",
                 work.name,
                 2])

from PyFoam.Applications.CaseReport import CaseReport
CaseReport(args=["--decomposition",
                 work.name])
```

- Most utilities are implemented by a class
  - Found in the PyFoam.Applications-sublibrary
  - Name is usually the utility-name without pyFoam and .py
- The parameter args is the command line we would use on the shell
- The command is executed immediately at construction

Setting up for a parallel run

Still using only 2 CPUs - scripts/prepareRun.py

```
1  from PyFoam.Execution.ParallelExecution import LAMMachine
   machine=LAMMachine(nr=2)
```

- LAMMachine is used to interface with MPI
  - It is named like that for historic reasons
- Also works with machine-files

Setting up the run

Ignaz only wants a progress indicator - scripts/prepareRun.py

```
from PyFoam.Execution.AnalyzedRunner import AnalyzedRunner
from PyFoam.LogAnalysis.FoamLogAnalyzer import FoamLogAnalyzer
from PyFoam.LogAnalysis.TimeLineAnalyzer import TimeLineAnalyzer
analyzer=FoamLogAnalyzer()
analyzer.time=0. # fix around a bug
analyzer.addAnalyzer("time",TimeLineAnalyzer(progress=True))
```

- xxxLineAnalyzers look at one line of output at a time
  - The TimeLineAnalyzer searches for an expression Time = x.xx and updates it's parents
  - The parameter progress makes sure that only the time is printed to the standard-output
- FoamLogAnalyzer takes care of one or more LineAnalyzers and feeds them the log-lines

Starting the actual run

Finally the run is started - scripts/prepareRun.py

```
print "Running"
theRun=AnalyzedRunner(analyzer,
                      argv=["rhoPisoAbscheidParcelFoam",
                        "-case",work.name],
                      silent=True,
                      lam=machine)
theRun.start()
```

- AnalyzedRunner is a sub-class of BasicRunner
  - Any LogAnalyzer can be added and will be handed the output for analysis
- The lam-parameter starts the run in parallel

Running it

Ignaz tries the script

```
> ./scripts/prepareRun.py cases/twoCavityChannel testSeparator 0.7 0.01
 PyFoam WARNING on line 152 of file /software/PyFoam/FoamInformation.py : <brk>
      <cont>1.5-dev is already being used
Running blockMesh
Decomposing
Case is decomposed for 2 processors
              |      0      1
-----------------------------
      Points |    906    918
       Faces |   1658   1682
       Cells |    402    408
-----------------------------
 abscheider1 |      0     15
 abscheider2 |     15      0
      bottom |     27     26
defaultFaces |    804    816
       inlet |      0     10
      outlet |     10      0
         top |     37     38
Running
 t =  0.02567
```

Last `git` of the day

... we'll miss it

Once more with feeling

```
> cd $HOME/pyFoamAdvanced
> git clone pyFoamAdvanced.git -b step5parameterVariation step5
> cd step5
```

Aiming for world domination

- Now Ignaz wants it all:
  - Run a parameter variation unattended
  - Have the data analyzed
    - And written to a file he can open with a spreadsheet-program
  - Get pictures of every run

The class that stares at goats

- First we write a class that inherits from `LogLineAnalyzer`
- The purpose of this class is to look at every line of the OpenFOAM™ output and
  - See what it can learn from it
  - Store that data
  - Give it back if asked to
- In this concrete case this means: every time a particle leaves through a patch its mass will be added
  - to the total mass
  - to the data entry of the patch

Setting up the class

Constructor - scripts/AbscheideAnalyzer.py

```
1  import re

3  from PyFoam.LogAnalysis.LogLineAnalyzer import LogLineAnalyzer

5  class AbscheideAnalyzer(LogLineAnalyzer):
       def __init__(self):
7          LogLineAnalyzer.__init__(self)

9          self.massLeft=0
           self.removed={}
11
           self.removedExpr=re.compile("Abgeschieden at (.+) with m: (.+) N: <brk>
               <cont> (.+) = total: (.+)")
13         self.massExpr=re.compile("Current mass in system                    = (.+)")
```

- Making it a sub-class of LogLineAnalyzer
- The module re is PYTHONs module for parsing regular expressions

Analyzing a line

These two methods are necessary - scripts/AbscheideAnalyzer.py

```
1     def doAnalysis(self,line):
          self.analyzeLine(line)
3
      def analyzeLine(self,line):
5         m=self.removedExpr.match(line)
          if m!=None:
7             name=m.groups()[0]
              mass=float(m.groups()[1])
9             nr=float(m.groups()[2])
              total=float(m.groups()[3])
11
              try:
13                self.removed[name]+=<brk>
                      <cont>total
              except KeyError:
15                self.removed[name] =<brk>
                      <cont>total

17        m=self.massExpr.match(line)
          if m!=None:
19            self.massLeft=float(m.<brk>
                  <cont>groups()[0])
```

- analyzeLine
  - Checks whether one of the regular expressions was matched
  - Updates the data according to the matches
- The try/except-construct makes sure that only field entries that are needed are added

Reporting

These two methods are Ignazs idea - scripts/AbscheideAnalyzer.py

```
1       def doReport(self):
            summe=self.massLeft
3           print "Mass left in system:",<brk>
                <cont>self.massLeft
            for k,v in self.removed.<brk>
                <cont>iteritems():
5               print "Removed by",k,":",v
                summe+=v
7           print "Total mass accounted <brk>
                <cont>for:",summe

9       def addCSVLine(self,csv):
            csv["mass left in system"]=<brk>
                <cont>self.massLeft
11          for k,v in self.removed.<brk>
                <cont>iteritems():
                csv[k]=v
13
        def addTimeListener(self,other):
15          pass
```

doReport Prints the summarized information

addCSVLine Add data to a CSVCollector (a convenience-class for generating a consistent csv-file

addTimeListener Desgin relict. Forget it

An analyzer

A class to call the LineAnalyzer - scripts/AbscheideAnalyzer.py

```
1  from PyFoam.LogAnalysis.FoamLogAnalyzer import FoamLogAnalyzer

3  class AbscheideLogAnalyzer(FoamLogAnalyzer):
       def __init__(self):
5          super(AbscheideLogAnalyzer,self).__init__()
           self.addAnalyzer("abscheid",AbscheideAnalyzer())
7
       def doReport(self):
9          self.getAnalyzer("abscheid").doReport()
```

- A wrapper for the AbscheideAnalyzer
- Only needed for the testing-code

Some code to test the class

Usually not used - scripts/AbscheideAnalyzer.py

```
1  import sys
   if __name__=='__main__':
3      log=sys.argv[1]

5      abscheid=AbscheideLogAnalyzer()
       from PyFoam.LogAnalysis.LogAnalyzerApplication import <brk>
           <cont>LogAnalyzerApplication
7      analyze=LogAnalyzerApplication(abscheid)
       analyze.run(log)
9
       abscheid.doReport()
```

Using the script

```
> ./scripts/AbscheideAnalyzer.py ../step1/testSeparator/PyFoamRunner.<brk>
      <cont>rhoPisoAbscheidParcelFoam.logfile
2  Mass left in system: 2.920114636e-05
   Removed by outlet : 0.00997079885178
4  Total mass accounted for: 0.00999999999814
```

Testing the waters

Ignaz creates two simple scripts:

prepareTheCase.py A lobotomized version of the script in the last chapter:

- Creates the case
- Does not run the solver

This script has to be run also to prepare a case for the parameter variation

runSolverAndAnalyze.py Runs the solver and prints the report from the analyzer

Using the two scripts

```
> ./scripts/prepareTheCase.py cases/twoCavityChannel testCase 0.01
Running blockMesh
> ./scripts/runSolverAndAnalyze.py testCase
Mass left in system: 0.01
Total mass accounted for: 0.01
```

Setting up stuff

Start of scripts/inletVelocityVariation.py

```
1  import re,sys
   from os import path
3
   sys.path.append(path.dirname(path.abspath(sys.argv[0])))
5
   from AbscheideAnalyzer import AbscheideAnalyzer
7
   from PyFoam.Execution.AnalyzedRunner import AnalyzedRunner
9  from PyFoam.Applications.ClearCase import ClearCase
   from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile
11 from PyFoam.Basics.CSVCollection import CSVCollection
   from PyFoam.Applications.PVSnapshot import PVSnapshot
13
   from PyFoam.FoamInformation import changeFoamVersion
15 changeFoamVersion("1.5-dev")

17 case=sys.argv[1]
   csv=CSVCollection(sys.argv[2]+".csv")
```

- Nothing new here

The actual variation

Second part and end of scripts/inletVelocityVariation.py

```
for v in [0.1, 0.5, 1, 1.5, 2, 3]:
    print "Velocity set to",v
    ClearCase(args=[case])
    uInit=ParsedParameterFile(path.join(case,"0","U"))
    uInit["boundaryField"]["inlet"]["value"].setUniform([v,0,0])
    uInit.writeFile()

    abscheid=AbscheideAnalyzer()

    run=AnalyzedRunner(abscheid,
                       argv=["rhoPisoAbscheidParcelFoam","-case",case],
                       silent=True)
    run.start()

    csv["Velocity"]=v
    abscheid.addCSVLine(csv)
    csv.write()

    abscheid.doReport()

    PVSnapshot(args=[case,
                     "--time=1",
                     "--time=1.5",
                     "--time=2",
                     "--time=3",
                     "--file-prefix=velocity=%g" % v])
```

- The only new thing is the loop

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Running the variation

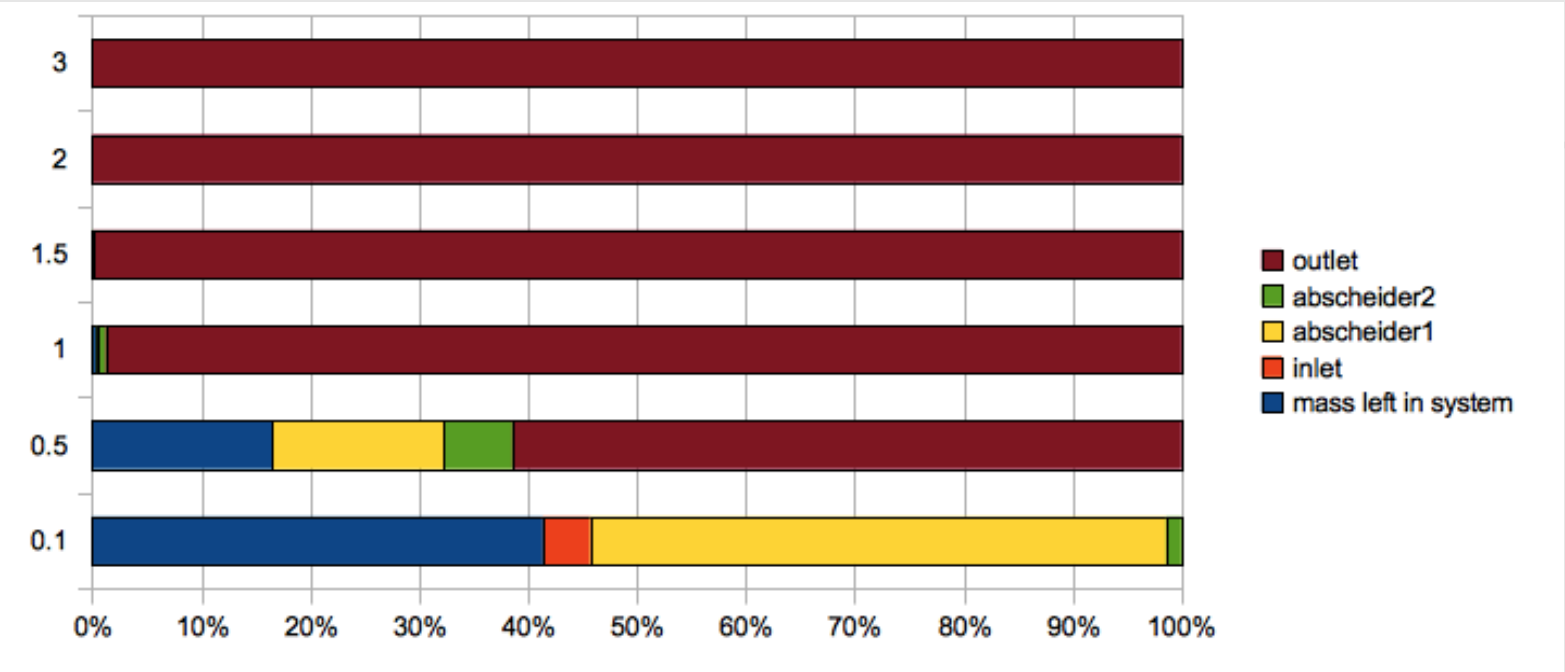
One command. 6 runs. One result file. 24 Pictures

```
> ./scripts/prepareTheCase.py cases/twoCavityChannel testVariation 0.005
Running blockMesh
> ./scripts/inletVelocityVariation.py testVariation variationData
```

- While this is running you can have a look at the data this will produce in `exampleResults/velTest.csv`
- Another remark: due to a bug in ParaView or `PVSnapshot` each iteration of the loop will produce multiple copies of the same pictures

The end result

The data from the CSV-file in one picture:



- Most variations were inefficient (no particles separated)
- Too slow is bad, too
- For low velocities the simulation time is too short (still mass in system)

Suggested exercises

- Try to find the optimum velocity (where the most mass comes to `abscheider2`)
  - Obviously extend the simulation time. But not too long. Suggestion: set it dependent on the inlet-velocity
- For the optimal velocity try varying the direction of the injector
- Check whether the cavities are to shallow
  - For this the mesh preparation would have to be pulled into the loop
- Try varying the upper channel wall
  - uniform (just the height)
  - making it narrower or wider in the middle

Acknowledgments

Thanks to

Holger Marschall for the idea for this tutorial

Astrid Mahrla for some pictures

Suggestions/Questions

Thanks for listening

- Questions?
- Suggestions for the library?

Bug reports at `http://sourceforge.net/apps/mantisbt/openfoam-extend/view_all_bug_page.php`