Real world PyFoam and swak4Foam

"Programming" an OpenFOAM-case

Bernhard F.W. Gschaider

Moscow
4. December 2015

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline I

Outline II

Outline

Outline

What this talk is about

- This talk will introduce two OpenSource-additions to OpenFOAM/Foam:
  - PyFoam
  - swak4Foam
- It will do so on the example of a real-world case
- Structure will follow the classic simulation 3-step:

and show how theses packages can help at each step

Warning

- There will be no movies in this presentations
- But around 100 slides
- It's OK if you want to leave now and go straight to the coffee
  - But I'd be sad

Warning

- There will be no movies in this presentations
- But around 100 slides
- It's OK if you want to leave now and go straight to the coffee
  - But I'd be sad

Warning

- There will be no movies in this presentations
- But around 100 slides
- It's OK if you want to leave now and go straight to the coffee
  - But I'd be sad

Warning

- There will be no movies in this presentations
- But around 100 slides
- It's OK if you want to leave now and go straight to the coffee
  - But I'd be sad

Warning

- There will be no movies in this presentations
- But around 100 slides
- It's OK if you want to leave now and go straight to the coffee
  - But I'd be sad

But first

- Introduction:
  - Who am I?
  - Where do I work?
  - What is PyFoam?
  - What is swak4Foam
  - What are we going to simulate
- I'll assume that you all know what OpenFOAM is

Bernhard Gschaider

- Author of
  - PyFoam
  - swak4Foam
- Administrator of
  - http://openfoamwiki.net
- Active in the OepnFOAM-community
- Employed at ICE Strömungsforschung
  - Most of my OpenSource stuff is "collateral damage" of customer projects done there

ICE Strömungsforschung

- Located in Leoben, Austria
- CFD Consulting
  - Development with OpenFOAM
  - Also using ClosedSource
- Customers in
  - Automotive industries
  - Petroleum
  - Manifacturing
  - Chemical processing
  - ...
- Active member of the OpenFOAM-community

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
    - in Training presentations at the OpenFOAM workshops

What is swak4Foam

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

swak4Foam stands for SWiss Army Knife for Foam. Like that knife it rarely is the best tool for any given task, but sometimes it is more convenient to get it out of your pocket than going to the tool-shed to get the chain-saw.

- It is the result of the merge of
  - funkySetFields
  - groovyBC
  - simpleFunctionObjects

  and has grown since
- The goal of swak4Foam is to make the use of C++ unnecessary
  - Even for complex boundary conditions etc

The core of swak4Foam

- At its heart swak4Foam is a collection of parsers (subroutines that read a string and interpret it) for expressions on OpenFOAM-types
  - fields
  - boundary fields
  - other (faceSet, cellZone etc)
- ... and a bunch of utilities, function-objects and boundary conditions that are built on it
- swak4foam tries to reduce the need for throwaway C++ programs for case setup and postprocessing

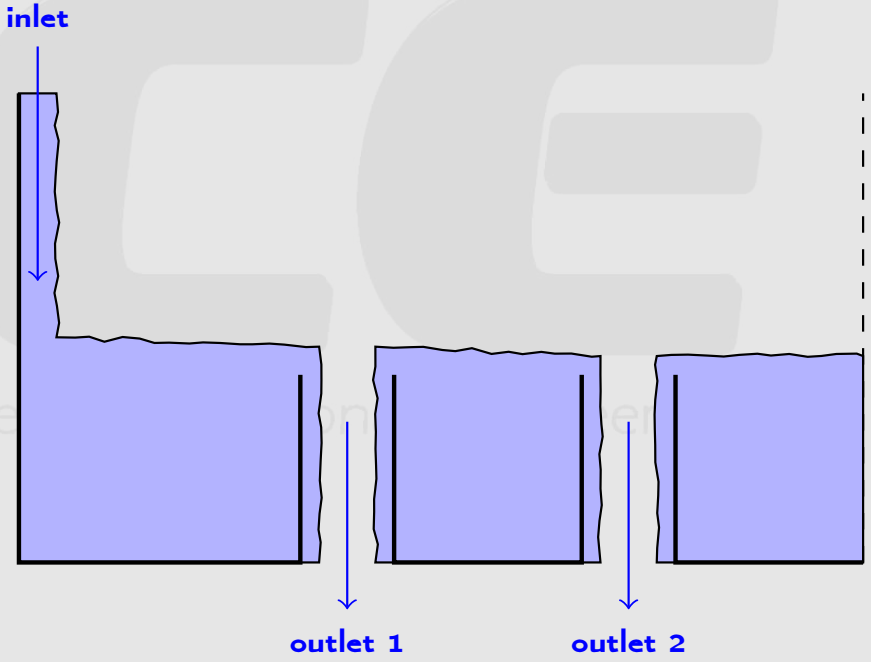
Outline

Real-life case

- Usually I use toy-cases for these presentations
  - The complexity of real world cases draws the attention from the real problems
- Not this time
- The case presented here is modelled on a real case
  - Geometries and parameters have been changed to protect the innocent
    - But the concepts remain valid
- We simulate a part of a polymer processing facility

Flow distributor for a polymere reactor

- Purpose of this thing is to
  - Receive a polymer flow from a previous stage in production
  - Distribute the polymer to the next stage



Goals of the simulation

- Given: Constant inflow
  - Looking for a pseudo-steady solution
- Find a way to distribute the flow as evenly as possible to the next stage
  - Investigate different arrangements of the pipes
  - Different geometries of the pipes
  - Other modifications that don't include
    - Modification of the outer geometry
    - Inflow

Local time-stepping (LTS)

- This is a pseudo-steady problem
  - Once converged the surface of the flow should not change anymore
- Reaching the steady-state with a pure transient solver takes a lot of time
  - Especially as the Courant-criterion limits the timestep
- For such cases OpenFOAM uses LTS (Local Time Stepping)
  - In each cell instead of a common timestep a local timestep is used
    - Calculated from the flow velocity so that the Courant-criterion is fulfilled
    - Smoothed to avoid jumps in the solution
  - Equations solved until a steady state is reached
    - Intermediate solutions are not physical

Outline

Outline

The `pyFoamPrepareCase.py`-utility

- To set up the case we use the `pyFoamprepareCase.py` utility
  - Helps automate the case-setup
- Does a number of things in a specific order:
  1. Clear the case
  2. Process templates
  3. Prepare the mesh
  4. Set initial conditions
- Working with this utility is like *programming the case*
  - More work has to be invested in the beginning
  - But afterwards automation is easy

The template format

- Templates are text files from which text files are generated
  - Using values that are inserted
- `PyFoam` comes with its own templating engine
  - Based on Python: calculations are done in the Python syntax
  - Control structures like `if` or `for` are enclosed in `<!--(` and `)-->`
    - Things like `if` or `for`
  - Expressions between `|-` and `-|` are evaluated and the result is inserted into the text file
  - Lines starting with `$$` are variable declarations
  - Everything else is passed to the result files

The parameter files

- The values for the templates are taken from parameter files
- Syntax of parameter files is the syntax of regular OpenFOAM-files
  - Including `#include` to pull in other files

Setting the time in controlDict

Template: controlDict.template

```
<!--(if solver=="LTSInterFoam")-->
endTime          |-ltsIterations-|;
<!--(else)-->
endTime          |-realTime-|;
<!--(end)-->
<!--(if solver=="LTSInterFoam")-->
deltaT           1;
<!--(else)-->
deltaT           |-deltaT-|;
<!--(end)-->
LTSInterFoam")-->
writeInterval␣␣␣␣␣␣␣␣␣␣|-int(ltsIterations/numberOfOutputs)-|;
<!--(else)-->
writeInterval␣␣␣␣␣␣␣␣␣␣|-float(realTime)/numberOfOutputs-|;
<!--(end)-->
```

The parameter file

Inserting these values into the template

```
solver LTSInterFoam;
ltsIterations 20000;
deltaT          0.01;
numberOfOutputs 100;
```

Result: controlDict

```
endTime          20000;
writeInterval          200;
deltaT           1;
```

Setting the time in controlDict

Template: controlDict.template

```
<!--(if solver=="LTSInterFoam")-->
endTime          |-ltsIterations-|;
<!--(else)-->
endTime          |-realTime-|;
<!--(end)-->
<!--(if solver=="LTSInterFoam")-->
deltaT           1;
<!--(else)-->
deltaT           |-deltaT-|;
<!--(end)-->
LTSInterFoam")-->
writeInterval␣␣␣␣␣␣␣␣␣␣|-int(ltsIterations/numberOfOutputs)-|;
<!--(else)-->
writeInterval␣␣␣␣␣␣␣␣␣␣|-float(realTime)/numberOfOutputs-|;
<!--(end)-->
```

The parameter file

Inserting these values into the template

```
solver LTSInterFoam;
ltsIterations 20000;
deltaT          0.01;
numberOfOutputs 100;
```

Result: controlDict

```
endTime          20000;
writeInterval          200;
deltaT           1;
```

Setting the time in controlDict

Template: controlDict.template

```
<!--(if solver=="LTSInterFoam")-->
endTime          |-ltsIterations-|;
<!--(else)-->
endTime          |-realTime-|;
<!--(end)-->
<!--(if solver=="LTSInterFoam")-->
deltaT           1;
<!--(else)-->
deltaT           |-deltaT-|;
<!--(end)-->
LTSInterFoam")-->
writeInterval␣␣␣␣␣␣␣␣␣␣|-int(ltsIterations/numberOfOutputs)-|;
<!--(else)-->
writeInterval␣␣␣␣␣␣␣␣␣␣|-float(realTime)/numberOfOutputs-|;
<!--(end)-->
```

The parameter file

Inserting these values into the template

```
solver LTSInterFoam;
ltsIterations 20000;
deltaT          0.01;
numberOfOutputs 100;
```

Result: controlDict

```
endTime          20000;
writeInterval          200;
deltaT           1;
```

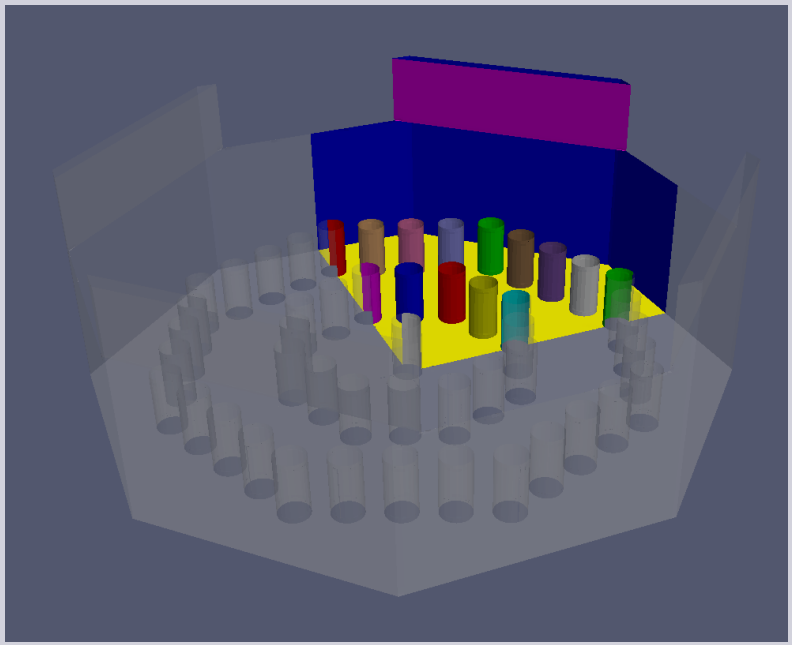
Outline

snappyHexMesh

- `snappyHexMesh` is one of the meshers that come with OpenFOAM
- Automatically generates a hex-dominant mesh
  - User *only* has to specify the boundaries . . . ideally
  - and the base mesh generated with `blockMesh` (the other mesher)
- Boundaries are specified in surface mesh files (we use STL)
- Additional parameters specify where to refine
- Mesher needs hints about the *feature edges*
  - Otherwise they will be "blunt"
- Everything is controlled from one text file

Using the symmetry

Whole geometry



Using the symmetry

- The geometry:
  - The overall geometry is a regular eight-side polygon
  - On four of the sides there are inlets
  - Pipes are aranged in a regular pattern
- All this means that we've only got to simulate one quarter of the whole geometry

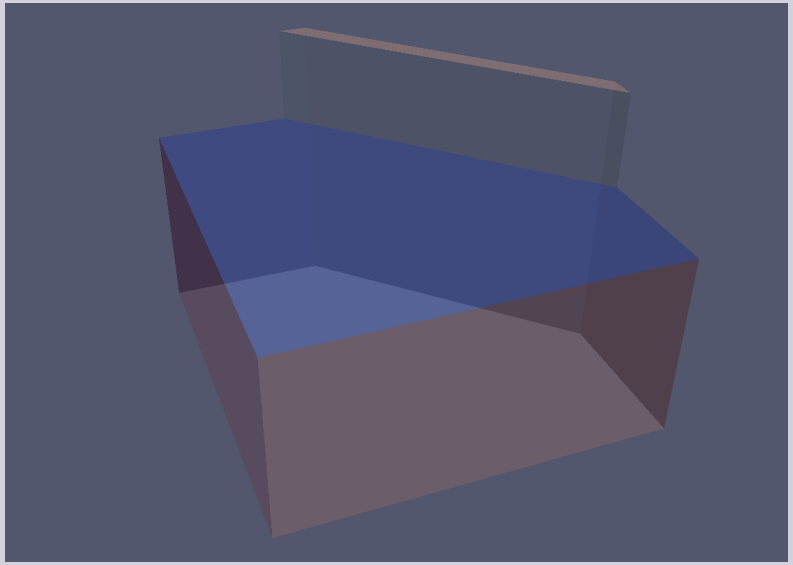
Outer geometry

Boundaries

The outer radius of this geometry is $1m$

- Inlet
- Wall
- Floor
- Symmetry
- Outlets will be set by the pipes

Outer geometry



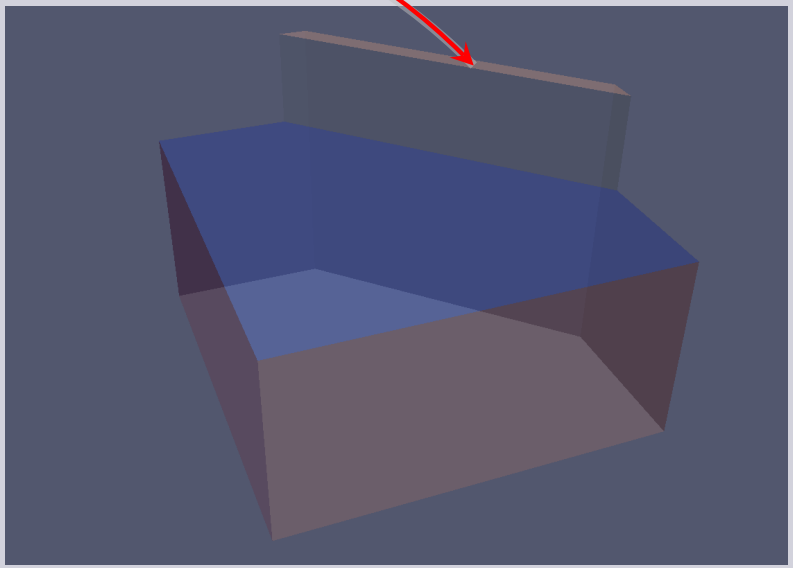
Outer geometry

Boundaries

The outer radius of this geometry is $1m$

- Inlet
- Wall
- Floor
- Symmetry
- Outlets will be set by the pipes

Outer geometry



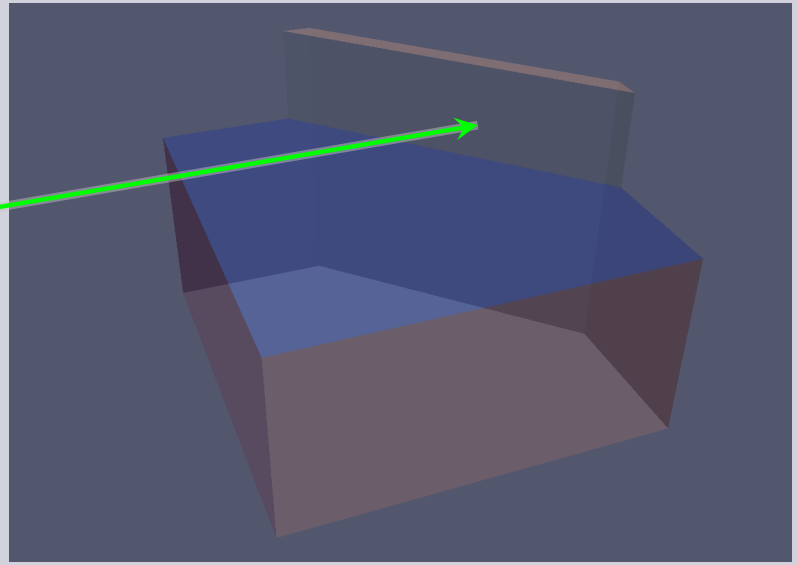
Outer geometry

Boundaries

The outer radius of this geometry is $1m$

- Inlet
- Wall
- Floor
- Symmetry
- Outlets will be set by the pipes

Outer geometry



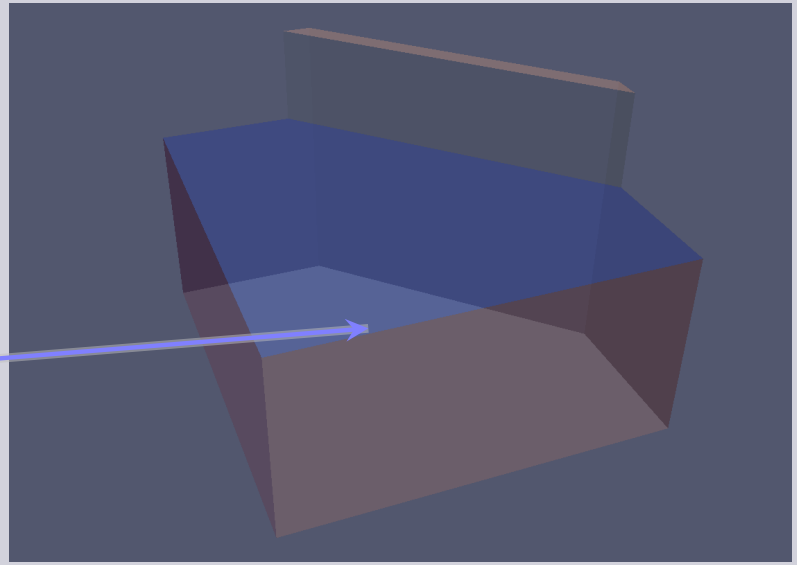
Outer geometry

Boundaries

The outer radius of this geometry is $1m$

- Inlet
- Wall
- Floor
- Symmetry
- Outlets will be set by the pipes

Outer geometry



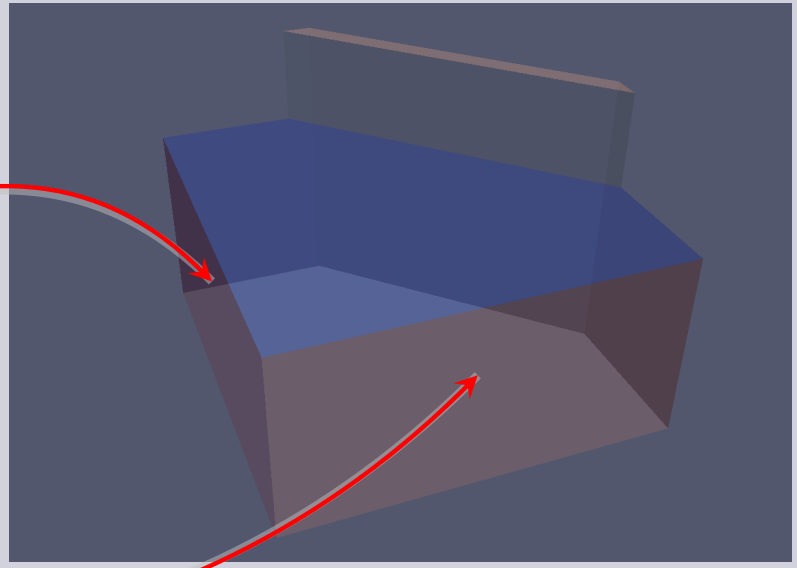
Outer geometry

Boundaries

The outer radius of this geometry is $1m$

- Inlet
- Wall
- Floor
- Symmetry
- Outlets will be set by the pipes

Outer geometry



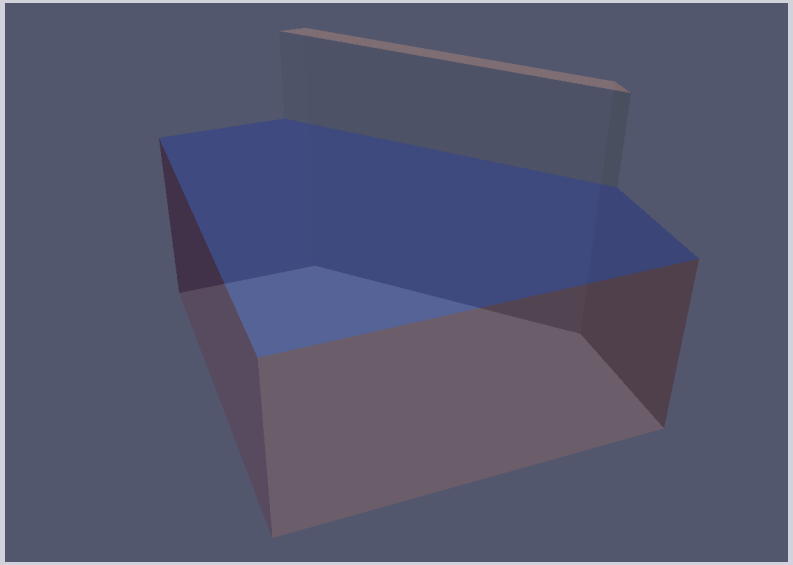
Outer geometry

Boundaries

The outer radius of this geometry is $1m$

- Inlet
- Wall
- Floor
- Symmetry
- Outlets will be set by the pipes

Outer geometry



Feature edges

Feature edges

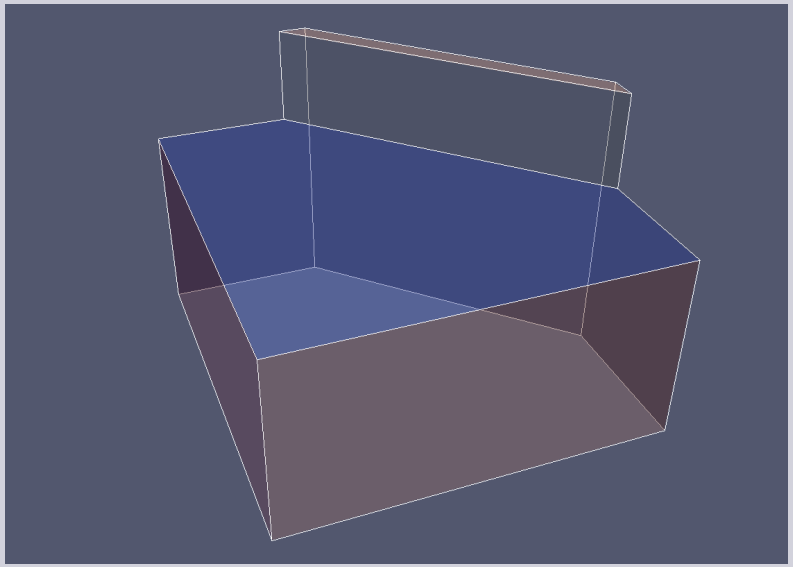
- Feature edges are detected by the `surfaceFeatureExtract`-utility
  - Controlled by a separate file
  - Needs names of surfaces and feature angles
- Feature edges are depicted by white lines in the picture

Feature edges

Feature edges

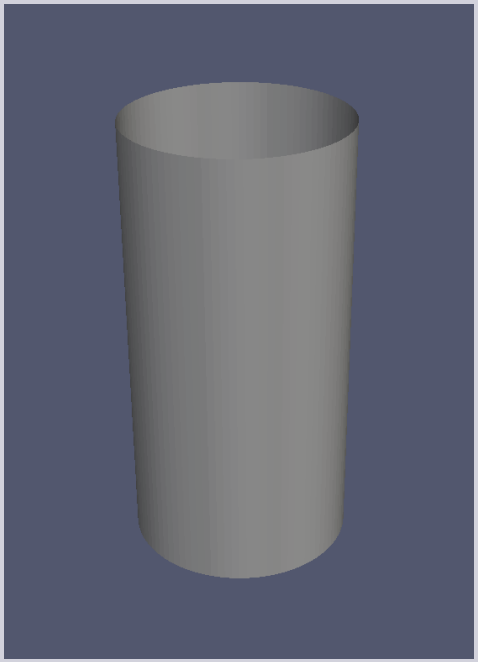
- Feature edges are detected by the `surfaceFeatureExtract`-utility
  - Controlled by a separate file
  - Needs names of surfaces and feature angles
- Feature edges are depicted by white lines in the picture

Edges on the geometry



Outflow pipes

The pipe



Outlet geometry

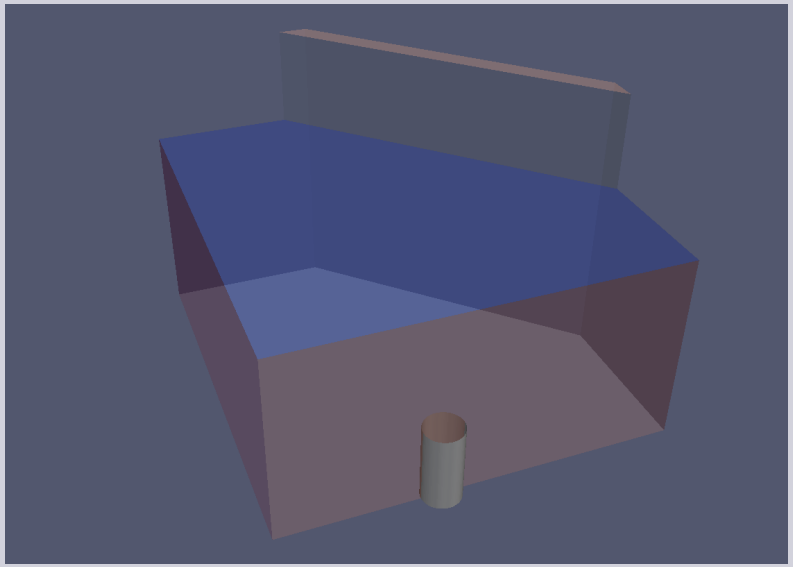
- This is the simplest possible outlet geometry
  - Height of the pipe is $20cm$
- The lower part intersects with the reactor
  - Polymer flows out
- A number of those is placed in the reactor
- We use just one STL
  - Centered at location $(0, 0)$
- Geometries with non-zero thickness of the wall are possible but have to be treated slightly different

Placing a pipe

Placing one pipe

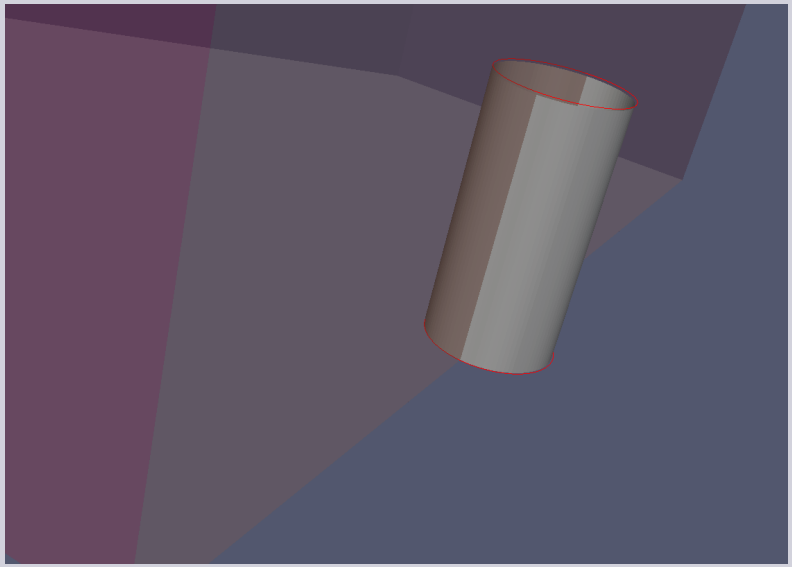
- Pipe geometry is shifted to a new location
  - There is a utility for this
- In our case it intersects with a symmetry plane

Pipe in the geometry



Feature edges of the cylinder

Edges of the cylinder



Every cylinder has its edges

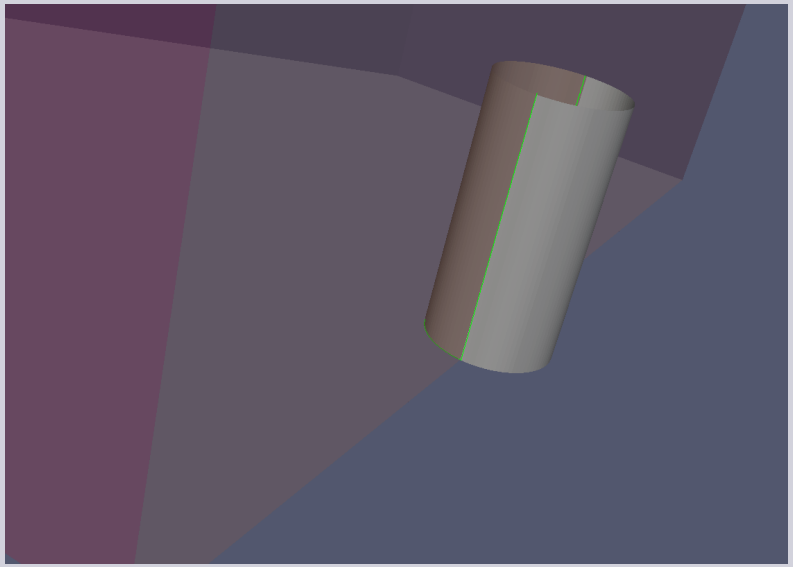
- Pipe surface has to be added to `surfaceFeatureExtractDict`
- Only the feature of the pipe itself is extracted
  - Red line

Intersection two geometries

Problem with boundary cylinders

- `snappyHexMesh` produces ugly cells where the pipe intersects the symmetry plane
  - Needs an additional hint
- There is a utility for that:
  - `surfaceBooleanFeatures` extracts the `difference` between the two STLs
  - Green lines in the picture

Intersection with outer geometry



Creating the outlets

- `snappyHexMesh` creates the geometry
  - With walls for the pipe
  - but the outlets are part of the "floor"
- Need two utilities to create the outlet
  - `topoSet` to identify the faces on the outlets
  - `createPatch` to create the actual patches from these sets
- Both utilities are controlled by text files
  - Operations have to be done separately for each outlet
- Whenever I say "text file" I mean "template file"

Other snappy settings

- Base mesh is rather coarse
  - No use to be accurate in the regions far from the surface
- Regions where interfaces are expected are refined
  - Liquid level plus/minus
    - approximately height of the pipes
  - Inside and around the pipes
- There are parameters for this
  - Zones are implemented in the `snappyHexMeshDict.template`

Looping over the pipes

snappyHexMeshDict.template

Specify feature edges once in the template file

```
    features
    (
<!--(for i,s in enumerate(einbauSpec))-->
$$ stlName="einbau%03d" % i
        {
            file "|-stlName-|.extendedFeatureEdgeMesh";
            level |-nFeatureLevel-|;
        }
  <!--(if intersectWithOuter)-->
        {
            file "outer_|-stlName-|_difference.extendedFeatureEdgeMesh";
            level |-nFeatureLevel-nIntersectDecrease-|;
        }
  <!--(end)-->
<!--(end)-->
```

snappyHexMeshDict

Gets generated for N pipes

```
features
(
    {
        file "einbau000.extendedFeatureEdgeMesh";
        level 5;
    }
    {
        file "outer_einbau000_difference.extendedFeatureEdgeMesh";
        level 5;
    }
    {
        file "einbau001.extendedFeatureEdgeMesh";
        level 5;
    }
```

Looping over the pipes

snappyHexMeshDict.template

Specify feature edges once in the template file

```
    features
    (
<!--(for i,s in enumerate(einbauSpec))-->
$$ stlName="einbau%03d" % i
        {
            file "|-stlName-|.extendedFeatureEdgeMesh";
            level |-nFeatureLevel-|;
        }
  <!--(if intersectWithOuter)-->
        {
            file "outer_|-stlName-|_difference.extendedFeatureEdgeMesh";
            level |-nFeatureLevel-nIntersectDecrease-|;
        }
  <!--(end)-->
<!--(end)-->
```

snappyHexMeshDict

Gets generated for N pipes

```
features
(
    {
        file "einbau000.extendedFeatureEdgeMesh";
        level 5;
    }
    {
        file "outer_einbau000_difference.extendedFeatureEdgeMesh";
        level 5;
    }
    {
        file "einbau001.extendedFeatureEdgeMesh";
        level 5;
    }
```

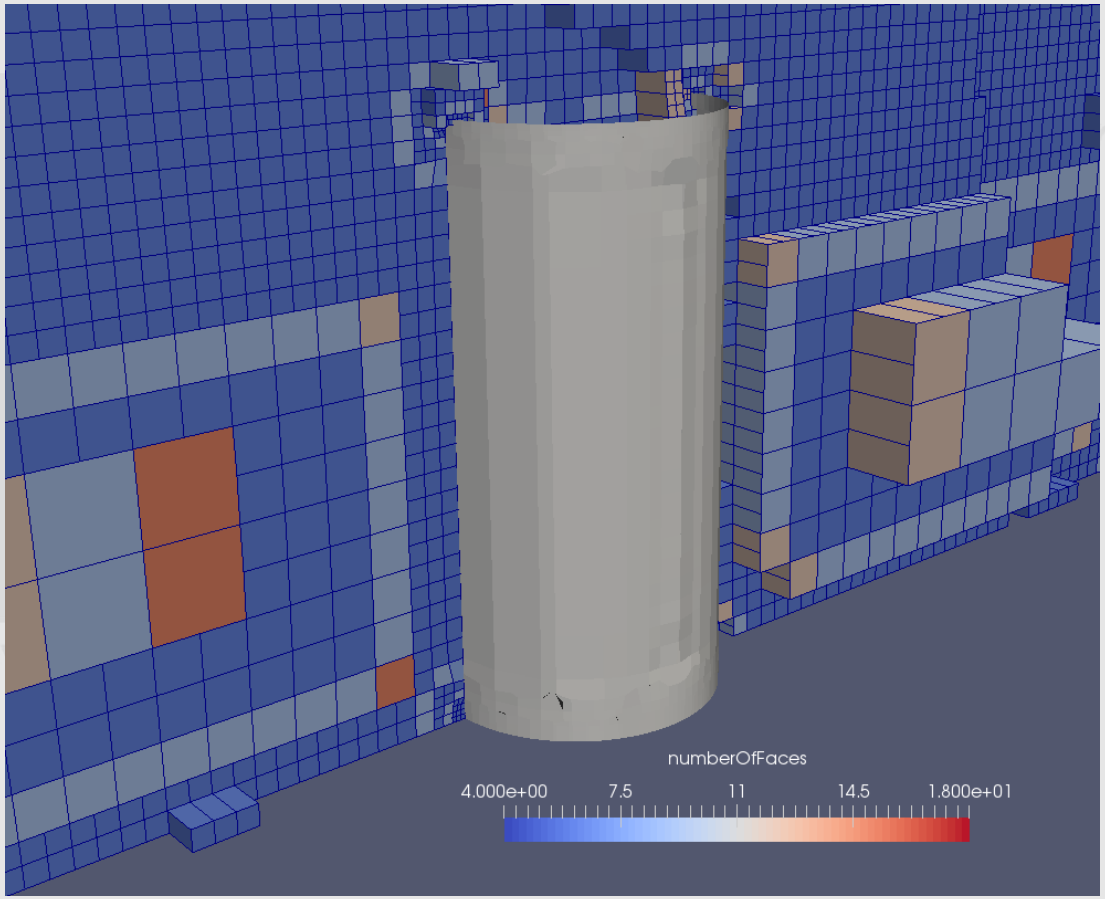
Annotated parameter values

- The file `default.parameters` can have a more elaborate syntax
  - Parameters can be organized in sections
  - Descriptive texts for parameters and sections
  - List of possible values (for instance: `solver` can only be `interFoam` or `LTSinterFoam`)
- `pyFoamPrepareCase.py` generates a structured document from these informations
  - Which parameter values were used (and which are changed from the default)
  - optionally as HTML or PDF

default.parameters

```
snappy {
    description "Settings for snappyHexMesh";
    values {
        nFeatureLevel {
            default 5;
            description "Refinement levels on feature edges";
        }
    }
```

Mesh for one one pipe



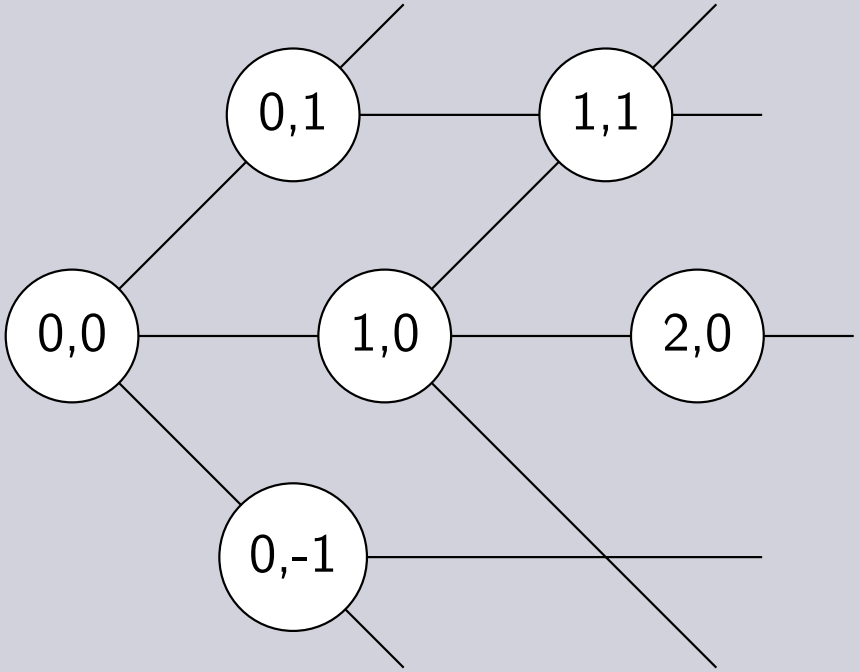
Outline

The coordinate system

Placing of the pipes

- Pipes are placed on a semi-regular grid (equal spacing)
  - One direction in a 45° angle to the other
  - Strange definition of the "negative" y-direction
- Coordinates could be calculated by hand
  - Lots of $\sqrt{2}$ in the calculation
- By providing a script `derivedParameters.py` `pyFoamPrepareCase.py` does the calculations for us

The grid



Specifying the positions

- Positions and geometries are specified in a list with
  - The STL to use
    - `default` means "take the STL from the variable `defaultSTL`" (this allows quick changes of the STL)
  - The coordinates
    - First two numbers are our "special coordinates"
    - Third number is how much the geometry should be shifted on the z-axis
- There is a separate list for geometries with non-zero thickness

Setting the default positions

achteckBarrieren.parameters

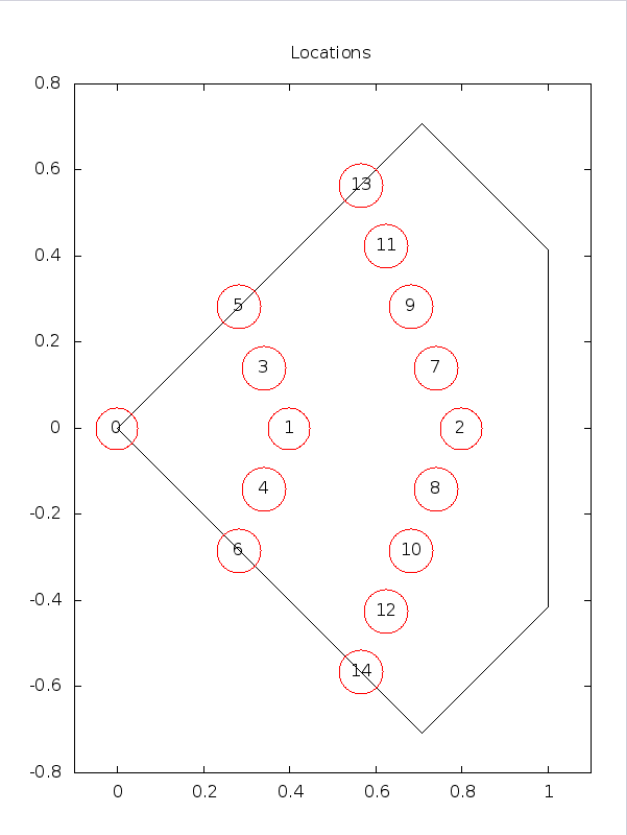
```
#include "achteckBase.parameters"

einbauSpec (
    (default    ( 0     0    0) )
    (default    ( 2     0    0) )
    (default    ( 4     0    0) )
    (default    ( 1     1    0) )
    (default    ( 1    -1    0) )
    (default    ( 0     2    0) )
    (default    ( 0    -2    0) )
    (default    ( 3     1    0) )
    (default    ( 3    -1    0) )
    (default    ( 2     2    0) )
    (default    ( 2    -2    0) )
    (default    ( 1     3    0) )
    (default    ( 1    -3    0) )
    (default    ( 0     4    0) )
    (default    ( 0    -4    0) )
);
```

Setting the default positions

achteckBarrieren.parameters

```
#include "achteckBase.parameters"

einbauSpec (
    (default    ( 0     0    0) )
    (default    ( 2     0    0) )
    (default    ( 4     0    0) )
    (default    ( 1     1    0) )
    (default    ( 1    -1    0) )
    (default    ( 0     2    0) )
    (default    ( 0    -2    0) )
    (default    ( 3     1    0) )
    (default    ( 3    -1    0) )
    (default    ( 2     2    0) )
    (default    ( 2    -2    0) )
    (default    ( 1     3    0) )
    (default    ( 1    -3    0) )
    (default    ( 0     4    0) )
    (default    ( 0    -4    0) )
);
```

The positions



Outline

groovyBC

- One of the oldest parts of `swak4Foam`
- Essentially a `mixed`-boundary condition where the important parts can be specified with expressions

  valueExpression expression of the values to set
  gradientExpression the gradient to set
  fractionExpression whether this face is a `value` (1) or a `gradient` (0) or something in between
- A list of `variables` can be specified to structure the expressions
- The expressions are evaluated separately for every face in the patch

Setting the massflow

U.template

```
    einlass|-einlassPatch-|
    {
<!--(if UInFromMassflow)-->
        type groovyBC;
        value uniform (0 0 0);
        variables (
            "totalArea=sum(area());"
            "URein=|-massFlowEinlass/<brk>
                <cont>densitySchmelze-|/<brk>
                <cont>totalArea;"
        );
        valueExpression "-normal()*URein";
<!--(else)-->
        type surfaceNormalFixedValue;
        refValue uniform |--UIn-|;
<!--(end)-->
    }
```

U

normal unit vector pointing outward for each face

area size of each face

sum sums up an expression

```
einlassUnten
{
    type groovyBC;
    value uniform (0 0 0);
    variables (
        "totalArea=sum(area());"
        "URein=0.00442477876106/totalArea;"
    );
    valueExpression "-normal()*URein";
}
```

Goals for the inital `alpha`-fields

- We don't want to start with an empty reactor
  - takes long to converge
  - the splashing of the incoming jet might "kill" the solver
- We don't want to start with a full reactor
  - takes long
  - we need an interface for gravity to "work"
- We try to guess a good initial solution
  - Liquid film at the inlet near the wall
  - Liquid level some centimeters above the upper edge of the tubes
  - Only a film of liquid on the inside walls of the tubes
- Usually we'd need a quite complicated C++-program to achieve this
  - And it would only be of use for this case

funkySetFields

- This is definitely the oldest part of swak4Foam
  - Basically swak started as a fusion of funky and groovy
- Allows quickly setting fields from the command line
  - For instance: we need a temperature field in Celsius for post-processing

```
> funkySetFields -latestTime -create -field TCelsius -expression "T-273.15"
```

- More flexibility is possible if one is using dictionary files
  - Expressions with variables
  - More than one expression in a row

The `caseSetup.sh`-script

- After the mesh setup `pyFoamPrepareCase.sh` executes this script
  - If present
    - Otherwise it will try to just execute `setFields`
  - Having to scripts allows selecting a phase
    - "Don't create the mesh - use the current one - but do everything else"
- In our case `caseSetup.sh` executes `funkySetFields`
  - With a dictionary file created from a template
    - The template uses information about the placement of the outlets
    - Parameters to specify the height of the liquid etc

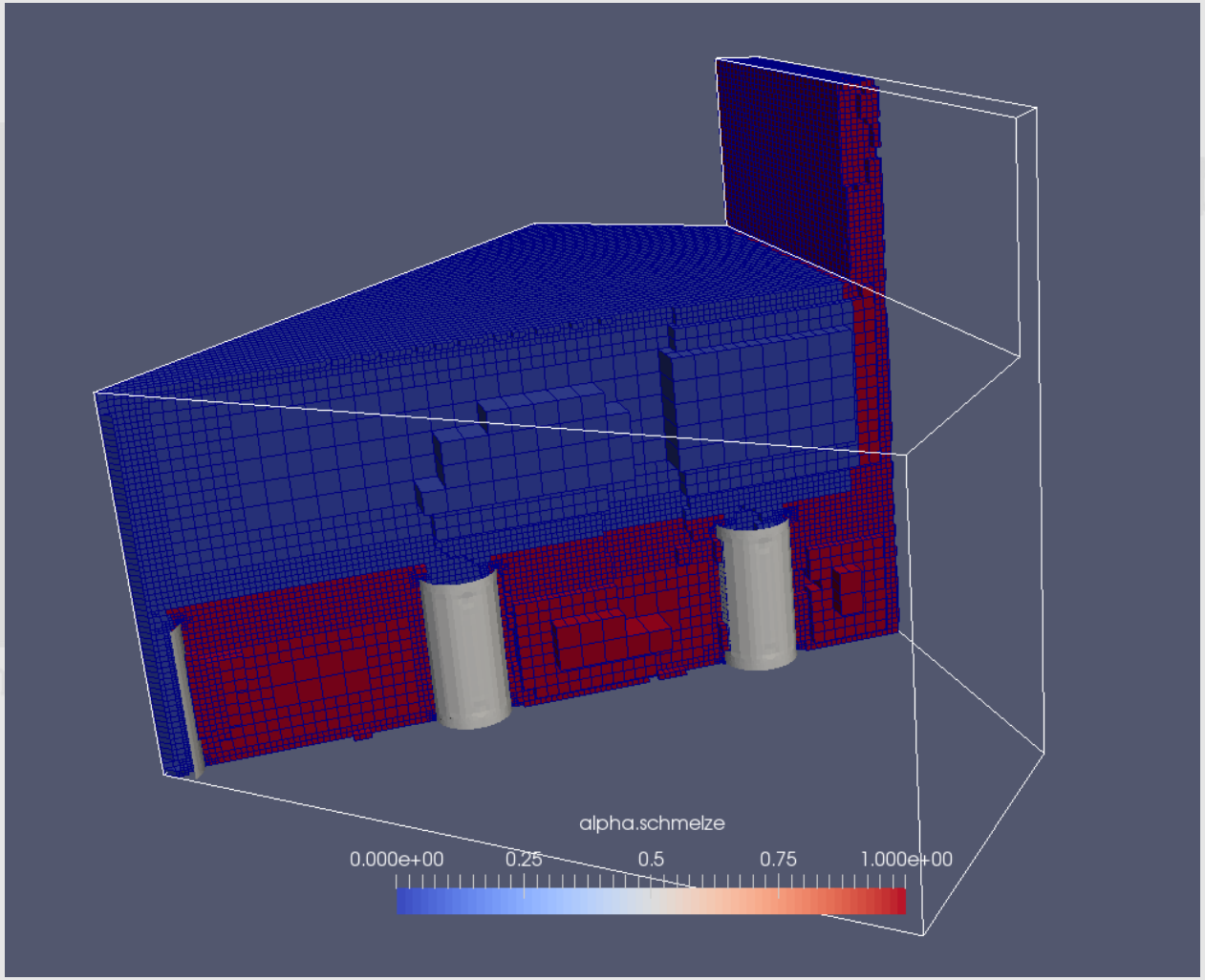
Setting the alpha field

funkySetFieldsDict.clearCylinders (from template)

We'll leave out the actual template

```
expressions (
    setFilm {
        field alpha.schmelze;
        keepPatches true;
        expression "1";
       condition "pos().x>0.94";
    }
    clearCylinder000
    {
        field alpha.schmelze;
        keepPatches true;
        expression "0";
        condition "(pow(pos().x-0.0,2)+pow(pos().y-0.0,2))<0.0016";
    }
    clearCylinder001
    {
        field alpha.schmelze;
        keepPatches true;
        expression "0";
        condition "(pow(pos().x-0.4,2)+pow(pos().y-0.0,2))<0.0016";
    }
```

Initial condition



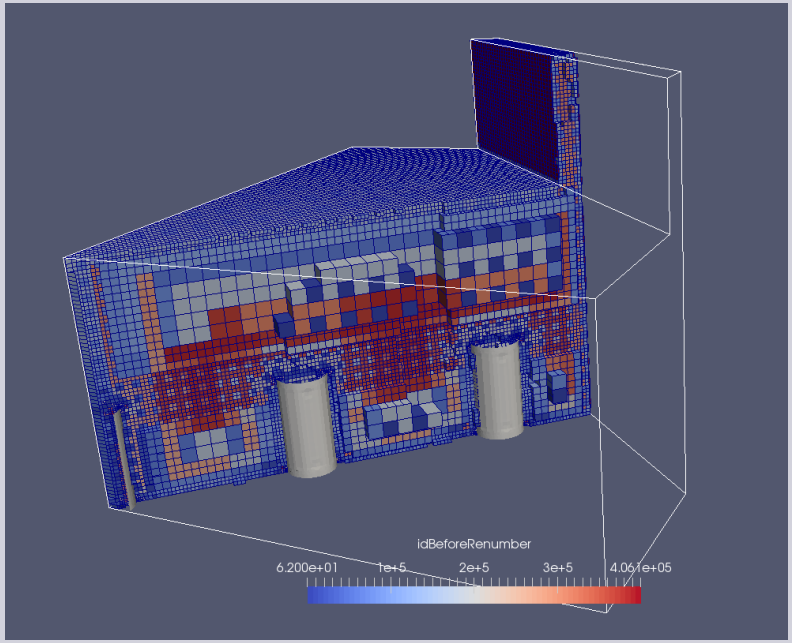
Outline

Other stuff in the `caseSetup.sh`-script

- Theoretically we're ready
  - Did I mention that the case has already been decomposed?
- But there are still things we want to:
  - Get fields that describe the mesh quality
  - Reorder the mesh to a lower bandwidth
    - Depending on the hardware and the original mesh this speed up calculations by up to 20%
  - In the following graphs cells that are next to each other in memory have a similar color

Reorder cells

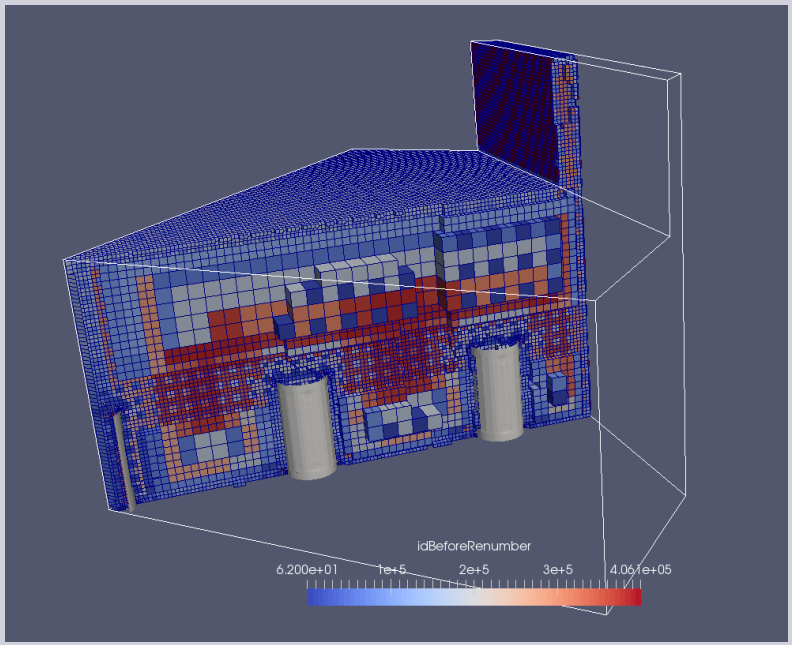
Original cell order



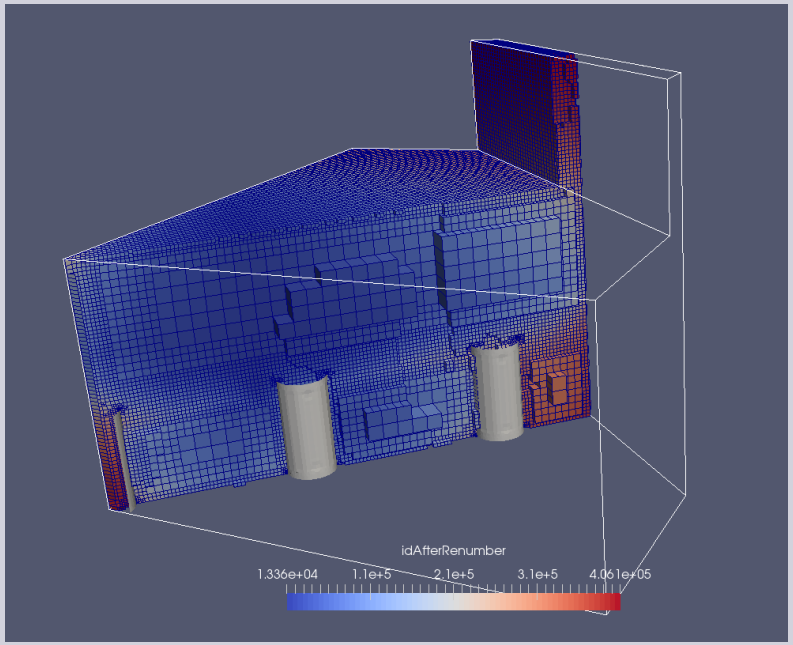
After `reorderMesh`

Reorder cells

Original cell order



After `reorderMesh`



How was this generated?

- The function `id()` in `funkySetFields` creates a field with the cell-id in each cell:

```
> funkySetFields -time 0 -create -field idBeforeRenumber -expression "id()"
```

- Next the `renumberMesh` utility is called

```
> renumberMesh -overwrite
```

- As the mesh is renumbered all the fields are renumbered too
- Now create the new IDs

```
> funkySetFields -time 0 -create -field idAfterRenumber -expression "id()"
```

- These three commands just have to be added to `caseSetup.sh`

Function plugins

- The syntax of `swak`-expressions offers a number of functions
  - Almost all functions that are available in OpenFOAM for fields
- Sometimes functions for special applications are needed:
  - Turbulence properties
  - Chemical reactions
  - Mesh quality
- As these functions are not needed by everyone (and some need a special "environment") these functions do not "pollute" the syntax
  - But they can be loaded via *function plugins*
  - and be used like regular functions

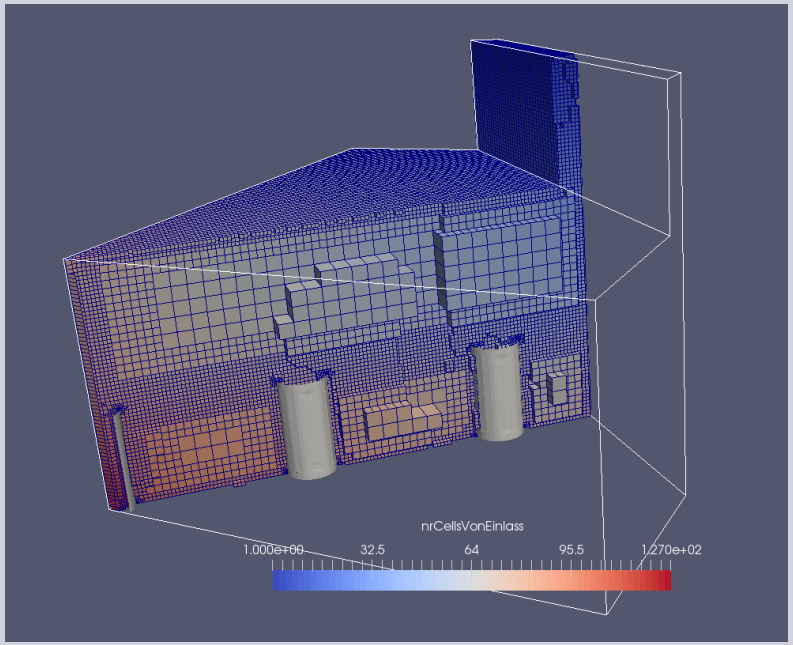
Cells from inlet

Estimate iteration number

- Theoretically the solution progresses one cell per iteration
  - So this number tells us how many iterations are needed till the inlet value "reaches" a cell
- For this the `MeshWave`-plugin was used

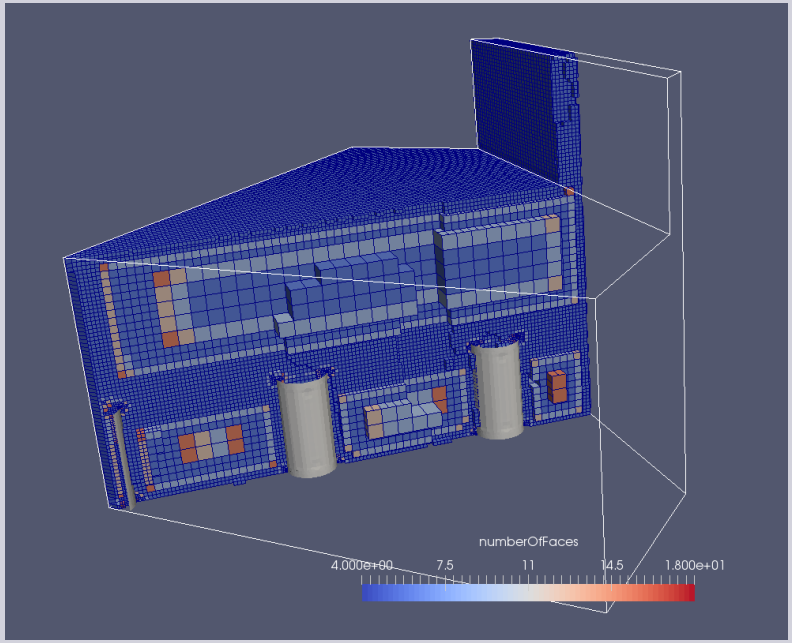
```
meshLayersFromPatch(einlass)
```

Cells from the inlet



Mesh quality

Number of faces per cell



Mesh quality functions

- The `MeshQuality` function plugin calculates various metrics (like `checkMesh`) of the mesh and writes them to fields
  - Orthogonality
  - Skewness
- In this picture the number of faces for each cell is printed

How the whole setup is done

- All the things we described are executed with one command

```
pyFoamPrepareCase.py . --parameter=achteckBarrieren.parameters
```

- Additional parameters can be overridden
  - By supplying a Python-dictionary with the values

Changing the mass flow

```
> pyFoamPrepareCase.py . --parameter=achteckBarrieren.parameters --values="{'<brk>
    <cont>massFlowEinlass':10}"
```

Outline

Outline

pyFoamRunner.py and pyFoamPlotRunner.py

- These are usually the first utilities of PyFoam people come in contact with
- They do the same things:
  - start an OpenFOAM-solver
  - capture its output
    - write it to screen
    - write it to a logfile
    - analyze it
  - ... some other things
- the difference is that the Plot-utility plots
- Typically we'd start our run with:

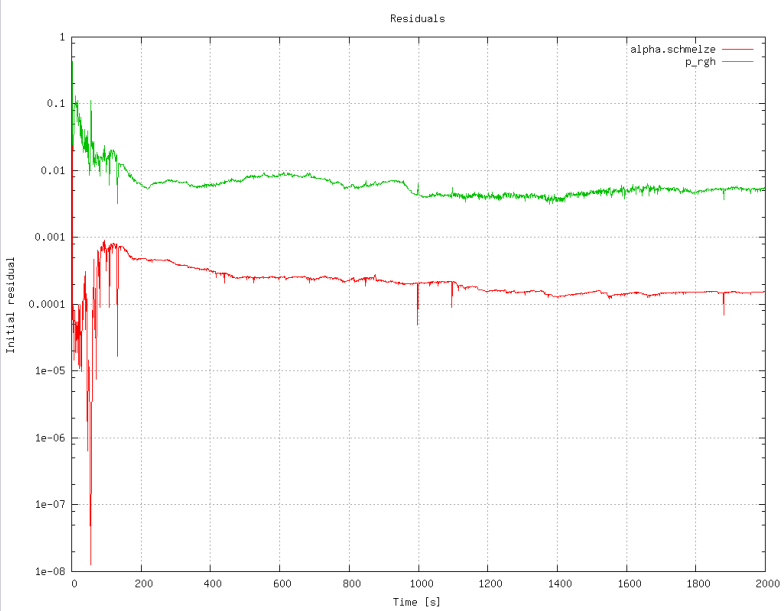
```
pyFoamPlotRunner.py --clear --progress --with-all interFoam
```

That also:

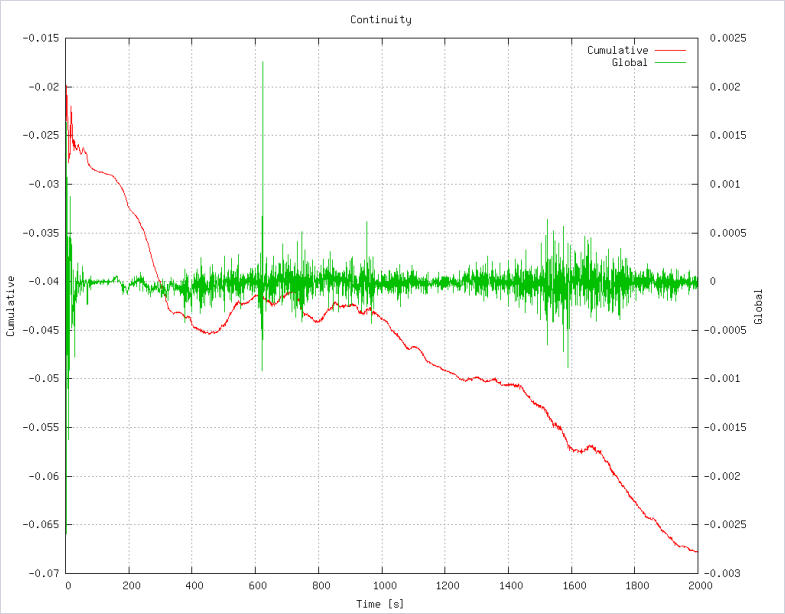
- removes time directories from a previous run
- only shows the current time (not the whole output)

Standard plots

Linear solver residuals

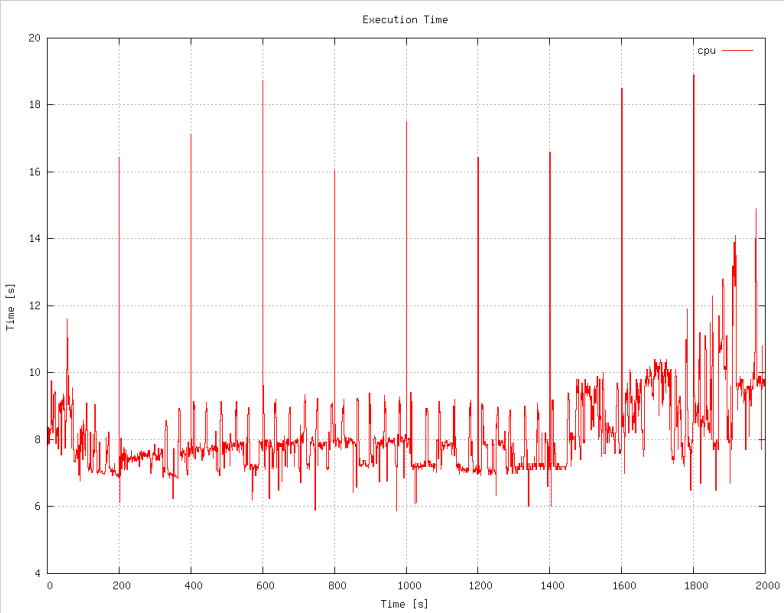


Continuity

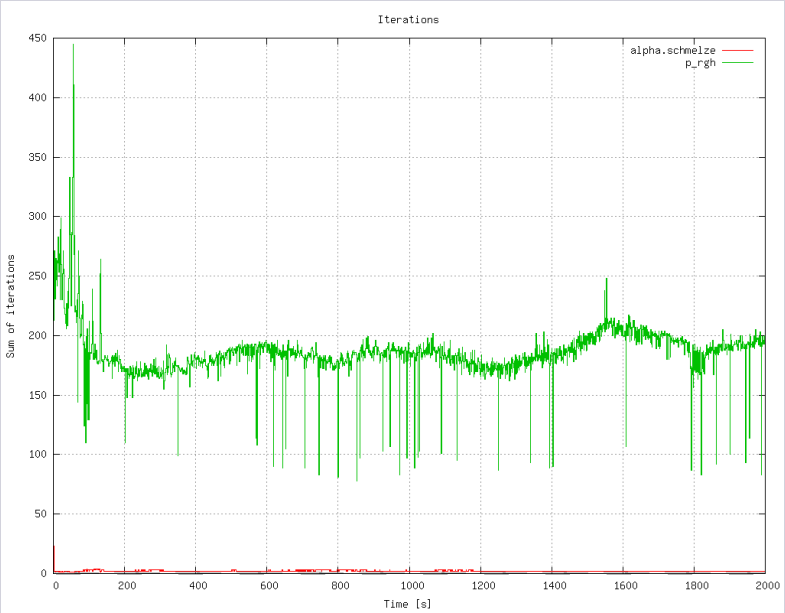


Additional plots

Execution time



Iterations of the linear solver



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Parallel support

- There is a utility for decomposing
  - it basically writes the `decomposeParDict` for you
- Decomposing a case to 5 processors:

```
pyFoamDecompose.py theCase 5
```

- The `Runner`-utilities have support for parallel execution

Running parallel

```
> pyFoamPlotRunner.py --autosense-parallel interFoam
```

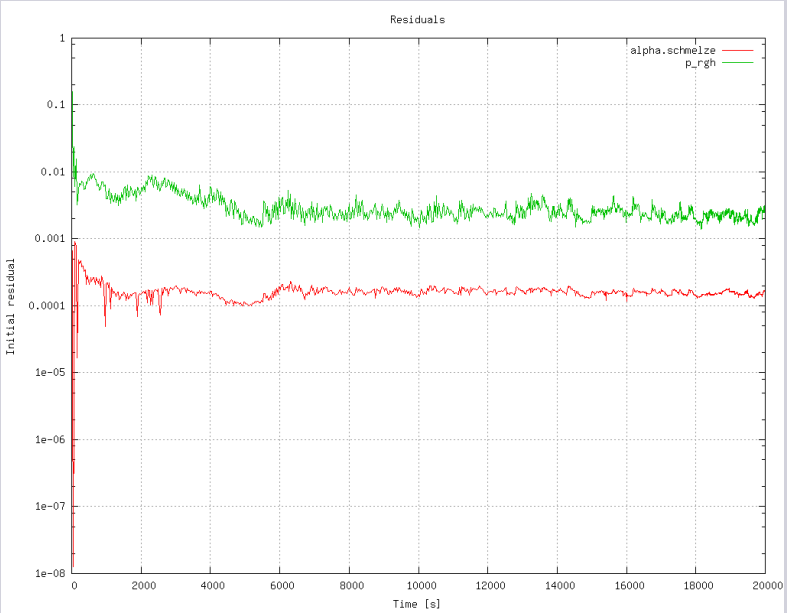
- Checks whether there are `processor`-directories
- If there are:
  - Prepends the appropriate `mpirun`-call
  - Appends `-parallel`

Is it converged?

Judging convergence from the residuals

- For this solver the residuals say nothing about the physical convergence of the flow
- After 6000 iterations there doesn't seem to be much change
  - But there are still changes
- So could we have stopped the run there?

Long term residuals



Outline

Average α

The `interFoam`-solvers print some information about the average volume fraction in the simulation

solver output

```
Time = 1

PIMPLE: iteration 1
smoothSolver:  Solving for alpha.schmelze, Initial residual = 0, Final residual = 0, No <brk>
      <cont>Iterations 1
;Phase-1 volume fraction = 0.451541;  Min(alpha.schmelze) = 0  Max(alpha.schmelze) = 1
MULES: Correcting alpha.schmelze
MULES: Correcting alpha.schmelze
Phase-1 volume fraction = 0.451541  Min(alpha.schmelze) = 0  Max(alpha.schmelze) = 1
DICPCG:  Solving for p_rgh, Initial residual = 1, Final residual = 0.0373154, No Iterations<brk>
      <cont> 17
Manipulated field U in 0 cells with the expression "mag(U)>100.0 ? Uunit*100.0 : U"
time step continuity errors : sum local = 1.31064, global = -0.0164968, cumulative = -0.01
```

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
- There is a number of cool "regular expression tester" (enter that in Google) applications on the web
  - One example: `http://regex101.com`

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 '[]\{\}'

Scanning for the alpha output

- The file `customRegexp` is automatically read by PyFoam
  - Used to analyze the output
- Every pattern is one dictionary (name is used for writing the data)

  theTitle title of the plot

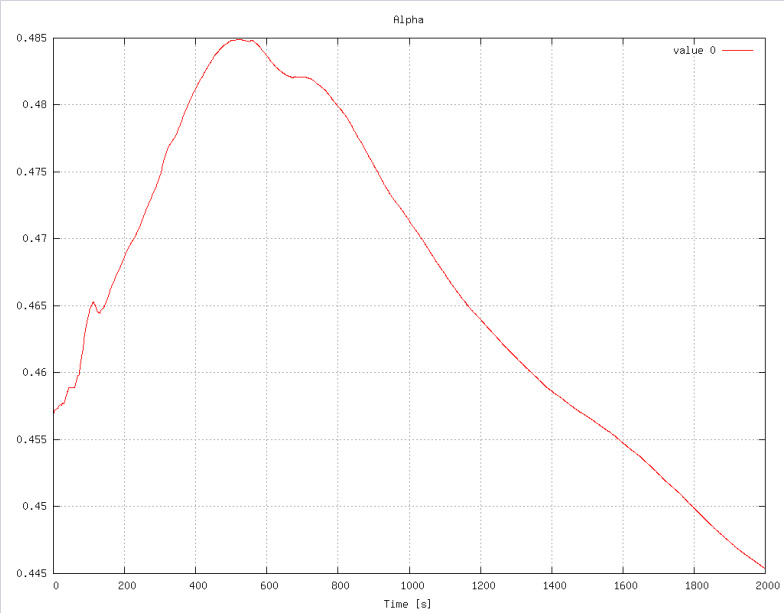
  expr the regular expression to look for
- Each *group* in the plot is assumed to be a data item
- `%f%` is a special PyFoam-abbreviation for the regular expression that matches a floating point number

customRegexp

```
alphaAverage {
             theTitle "Alpha";
             expr "Phase-1 volume fraction = ;(%f%);";
}
```

Is α converged?

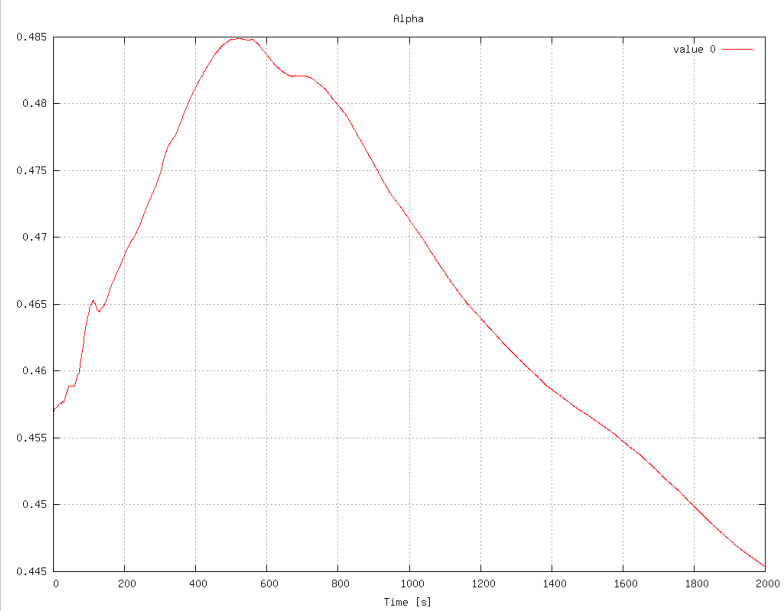
Now it isn't



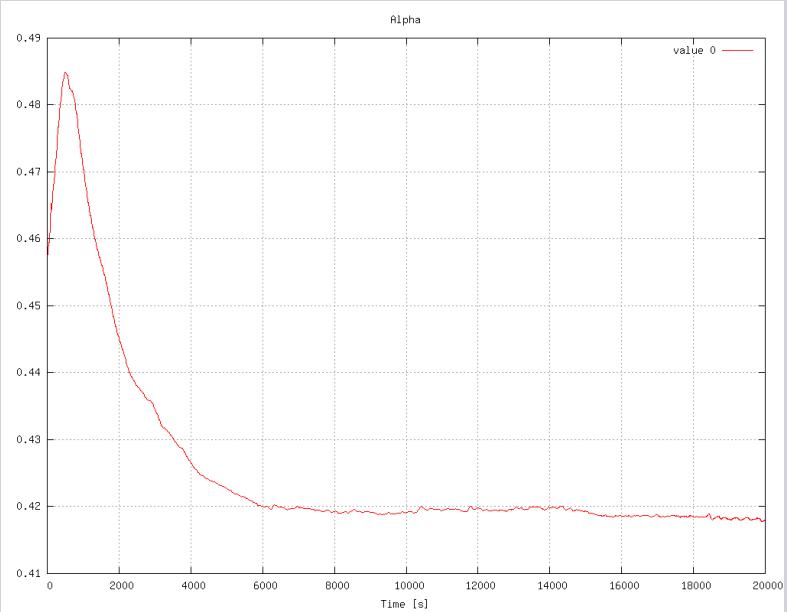
Now it may be

Is α converged?

Now it isn't



Now it may be



Producing our own output

- Most of the time the output the solver gives us isn't enough
- We need output that answers our questions
- To get that output we use function objects
  - The function objects prints the answers to the screen
  - PyFoam looks for it and plots it
- Most function-objects also write their results to the `postProcessing`-directory

Getting minimum and maximum

simpleFunctionObjects

- Collection of function objects that don't parse expressions
  - also an old part of swak
- volumeMinMax "only" prints the minimum and the maximum of a list of fields
  - verbose means that it writes its results to the terminal

controlDict

```
extremes {
    type volumeMinMax;
    fields (
        alpha.schmelze
        U
        p_rgh
    );
    verbose true;
    outputControlMode timeStep;
    outputInterval 1;
}
```

Plotting the extremes

Output and scan

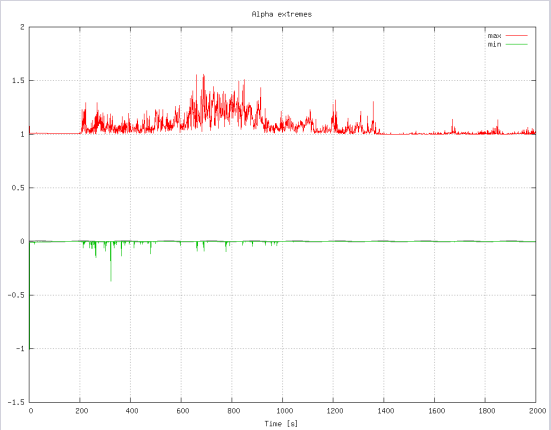
This output

```
Range of alpha.schmelze [ ;-0.002; , ;1.001; ] [0 0 0 0 0 <brk>
      <cont>0 0]
Range of p_rgh [ -68.8248 , 7918.26 ] [1 -1 -2 0 0 0 0]
Range of U [ (-1.51871 -2.8473 -1.73918) , (1.70397 <brk>
      <cont>2.87548 2.45731) ] [0 1 -1 0 0 0 0]
```

is caught by

```
alphaExtremes {
    theTitle "Alpha extremes";
    titles (
        min
        max
    );
    expr "Range of alpha.schmelze \[ ;(%f%); , ;(%f%); \]"<brk>
          <cont>;
}
```

Overshooting alpha



Desperate measures

- We see that the volume fractions are outside the range $[0, 1]$ that is physical possible
  - This is not uncommon during the start of the simulation
- It can make the solver crash
  - Can be avoided with smaller time-steps
    - But this makes the solution take longer
- The brutal method: reset all outside cells to the proper range
  - This should only be a temporary fix for the startup period
    - If this is necessary through the whole simulation we have a problem
- The `manipulateField` function object can do this

Clipping the fields

controlDict.template

```
<!--(if clipFields)-->
    clipAlpha {
        type manipulateField;
        fieldName alpha.schmelze;
        aliases {
            aSchmelze alpha.schmelze;
        }
        expression "aSchmelze > |-1+clipAlphaTolerance-| ? |-1+clipAlphaTolerance-| : (aSchmelze<|--<brk>
              <cont>clipAlphaTolerance-|) ? |--clipAlphaTolerance-| : aSchmelze";
        mask "(aSchmelze > |-1+clipAlphaTolerance-|) || (aSchmelze<|--clipAlphaTolerance-|)";
    }
```

controlDict

```
clipAlpha {
    type manipulateField;
    fieldName alpha.schmelze;
    aliases {
        aSchmelze alpha.schmelze;
    }
    expression "aSchmelze > 1.01 ? 1.01 : (aSchmelze<-0.01) ? -0.01 : aSchmelze";
    mask "(aSchmelze > 1.01) || (aSchmelze<-0.01)";
}
```

Clipping the fields

controlDict.template

```
<!--(if clipFields)-->
    clipAlpha {
        type manipulateField;
        fieldName alpha.schmelze;
        aliases {
            aSchmelze alpha.schmelze;
        }
        expression "aSchmelze␣>␣|-1+clipAlphaTolerance-|␣?␣|-1+clipAlphaTolerance-|␣:␣(aSchmelze<|--<brk>
              <cont>clipAlphaTolerance-|)␣?␣|--clipAlphaTolerance-|␣:␣aSchmelze";
        mask "(aSchmelze␣>␣|-1+clipAlphaTolerance-|)␣||␣(aSchmelze<|--clipAlphaTolerance-|)";
    }
```

controlDict

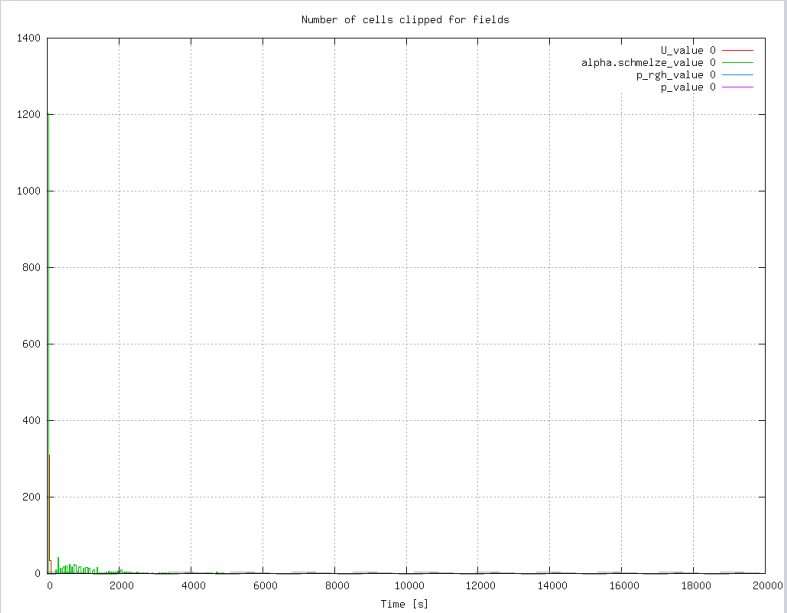
```
clipAlpha {
    type manipulateField;
    fieldName alpha.schmelze;
    aliases {
        aSchmelze alpha.schmelze;
    }
    expression "aSchmelze␣>␣1.01␣?␣1.01␣:␣(aSchmelze<-0.01)␣?␣-0.01␣:␣aSchmelze";
    mask "(aSchmelze␣>␣1.01)␣||␣(aSchmelze<-0.01)";
}
```

Clipped cells

How often are we clipping

- Cells are clipped during the startup phase
- In the end no clipping necessary
  - This means it is naughty but acceptable
    - Read: it validates the approach

Only clipping at the start



Porous term for the velocity (unused)

controlDict.template

This is an example for the `expressionField` function object that creates a new field

```
<!--(if dampVelocityPorous)-->
    UResistance {
        type expressionField;
        fieldName UResistance;
        variables (
            "magU=mag(U);"
            "resNew=|-resistivityMax-|*(magU>|-porousLowerUThres+porousUTransitionRegion-|␣?␣1␣:␣(magU<brk>
                  <cont><|-porousLowerUThres-|␣?␣0␣:␣((magU-|-porousLowerUThres-|)/|-<brk>
                  <cont>porousUTransitionRegion-|)));"
            "resOld=|-max(0,1-resistivityRelax)-|*resOld+|-min(1,resistivityRelax)-|*resNew;"
        );
        storedVariables (
            {
                name resOld;
                initialValue "0";
            }
        );
        expression "resOld";
        autowrite true;
    }
<!--(end)-->
```

Would have been used in a `fvOption` (didn't improve things)

Different velocities

- There are different velocities in our simulation

  velocity of our fluid this is what really interests us
  velocity of the air higher. Not so interesting for us
  velocity in the interface zone here sometimes the divergence starts
- Plotting these velocities helps us to judge the simulation
- We'll use the function object `swakExpression` for this
  - Evaluates expressions on fields, patches, zones ...
    - entry `valueType` selects which one is used

Calculating arbitrary expressions

controlDict

```
magUValuesOverall {
    type swakExpression;
    valueType internalField;
    verbose true;
    outputControlMode timeStep;
    outputInterval 1;
    expression "mag(U)";
    accumulations (
        weightedAverage
        weightedQuantile0.99
        max
    );
}
```

Calculating and summarizing

- `expression` is evaluated
  - `mag` calculates the length of a vector
- `accumulations` describes how to calculate a single value from the expression

max this is obvious

weightedAverage the volume weighted average of the value

weightedQuantile0.99 The value for which 99% of the volume has a smaller value

Other velocities

controlDict

```
magUValuesFluid {
    $magUValuesOverall;
    aliases {
        aSchmelze alpha.schmelze;
    }
    expression "aSchmelze>0.5 ? mag(U) : 0"<brk>
          <cont>;
}
magUValuesAir {
    $magUValuesFluid;
    expression "aSchmelze<0.5 ? mag(U) : 0"<brk>
          <cont>;
}
magUValuesMixed {
    $magUValuesFluid;
    expression "(aSchmelze>0.1 && aSchmelze<brk>
          <cont><0.9) ? mag(U) : 0";
}
```

Getting all in one plot

- Velocities in the different phases
  - Using thresholds to determine the phase
- All the velocities are written to 4 different lines in the output
  - But we want them in one plot

Dynamic plotting

- This allows collecting similar output into one plot
  - *Similar* means: looks the same except for the name
- Selected by `type dynamic`
- One regular expression group is the name
  - Selected by the entry `idNr`
- The list `titles` is common for all custom plots
  - Labels the curves get in the legend
- There is another way to append lines to other plots
  - `type slave`
    - A `master` plot has to be specified

Output and scanning expression

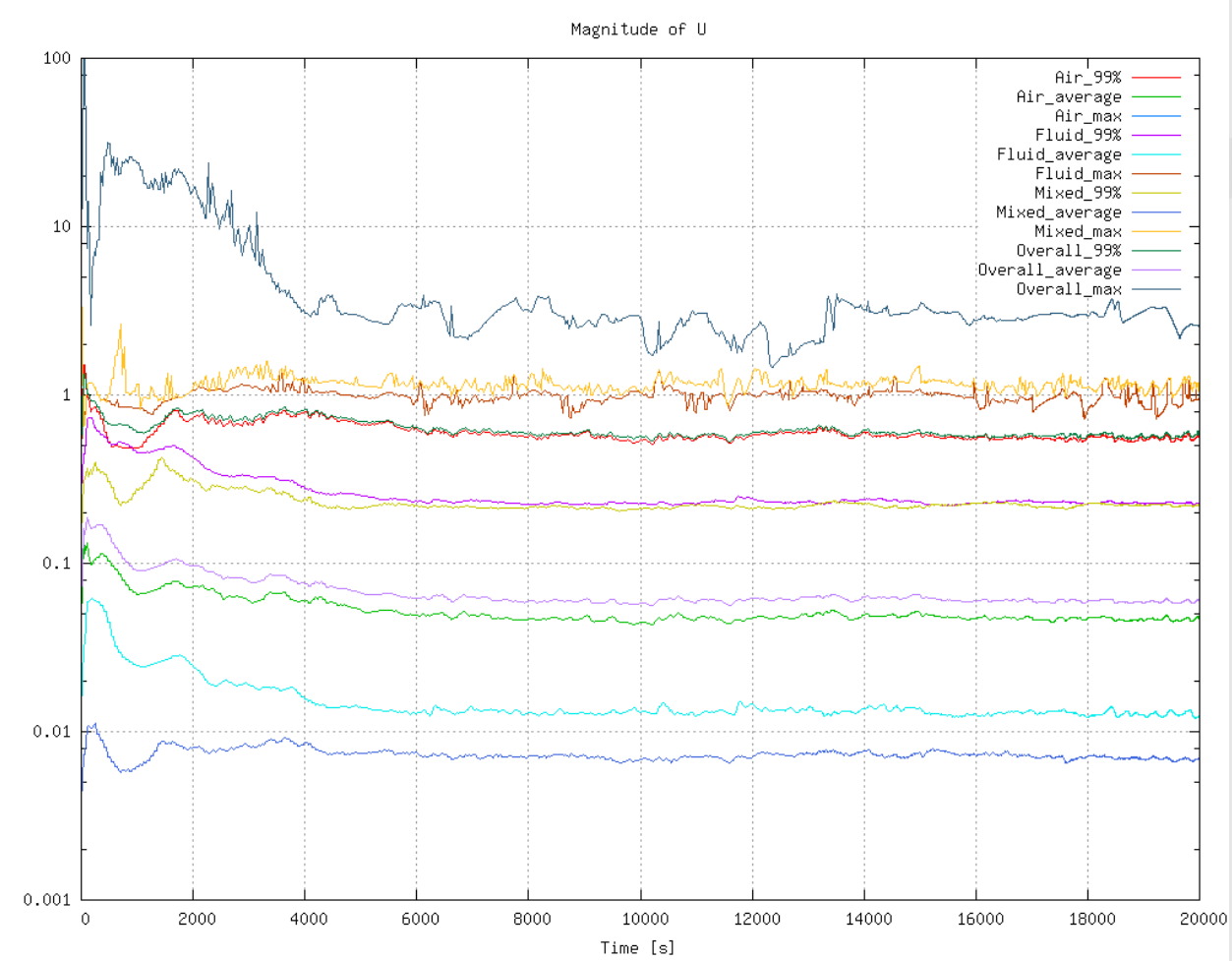
Output

```
Expression magUValues;Overall; :  weightedAverage=0.0633709 weightedQuantile0.99=0.479547 max=30.6012
Expression magUValues;Fluid; :  weightedAverage=0.0136898 weightedQuantile0.99=0.263334 max=0.460643
Expression magUValues;Air; :  weightedAverage=0.0496811 weightedQuantile0.99=0.474317 max=30.6012
Expression magUValues;Mixed; :  weightedAverage=0.00392488 weightedQuantile0.99=0.145665 max=1.07427
```

customRegexp

```
magUValues {
    type dynamic;
    theTitle "Magnitude of U";
    logscale true;
    idNr 1;
    expr "Expression magUValues;(.+); :  weightedAverage=(%f%) weightedQuantile0.99=(%f%) max=(%f%)";
    titles (
        average
        "99%"
        max
    );
}
```

Velocities


Magnitude of U
100
10
1
0.1
0.01
0.001
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time [s]
Air_99%
Air_average
Air_max
Fluid_99%
Fluid_average
Fluid_max
Mixed_99%
Mixed_average
Mixed_max
Overall_99%
Overall_average
Overall_max

Using the data

- `--hardcopy` generates bitmaps of the plots
- But sometimes these are not enough (they were for this presentation)
- `swak4Foam` writes to plain text files
  - Can be read by many utilities
- PyFoam writes to a special ("pickled") file
  - Can be extracted with utilities
  - Optionally plain text files can be written
- PyFoam also has utilities to convert these plain text files to `CSV` or even `Excel` files
  - Including filtering
  - And selection

Outline

What the cluster support does

- Based on a python class `ClusterJob`
  - There is a sub-class `PrepareCaseJob` that uses the machinery of `pyFoamPrepareCase.py` for setting up the case
- What it does
  1. Copy the essential files from a template case (clone)
     - to a new directory with a unique name
  2. Sets up the case
  3. Decomposes the case (gets the number of processors from the cluster engine)
  4. Runs the case
  5. Reconstructs it
- The script reads command line parameters and translates them

The cluster script

```
#!/opt/rocks/python/python
#
#$ -cwd
#$ -j y
#$ -S /opt/python/bin/python
#$ -m be
# #$ -pe mpi 2
#

from PyFoam.Infrastructure.ClusterJob import PrepareCaseJob
from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile

template=sys.argv[1]
suffix=sys.argv[2]
parameters=sys.argv[3]
arguments=sys.argv[4:]

argString=""
if len(arguments)>0:
    argString="_parameters="+"_".join(["%s=%s"%i for i in zip(arguments[::2],arguments[1::2])])

class Ueberlauf(PrepareCaseJob):
    def __init__(self):
        PrepareCaseJob.__init__(self,
                                "Ueberlauf_"+path.basename(path.abspath(template))+"_"
                                +parameters+argString+"_"+suffix,
                                "LTSInterFoam",
                                parameters,
                                arguments,
                                template=template,
                                steady=False,
                                cloneParameters=["--no-vcs"],
                                autoParallel=False,
                                foamVersion="2.3.1")

Ueberlauf().doIt()
```

Testing the script

- There is a utility to test these scripts locally
  - Emulates the environment on the cluster
  - The script runs the way it would on the cluster

```
> pyFoamClusterTester.py runUhdeUeberlauf.py templateCase test achteckBarrieren.parameters
```

- Test the case in parallel

```
> pyFoamClusterTester.py --procnr=2 runUhdeUeberlauf.py templateCase 2cpu achteckBarrieren.<brk>
    <cont>parameters
```

- Additional parameters

```
> pyFoamClusterTester.py runUhdeUeberlauf.py templateCase fast achteckBarrieren.parameters <brk>
    <cont>massFlowEinlass 10
```

Running it

- Actually running the script depends on the cluster
  - Currently only SGE supported

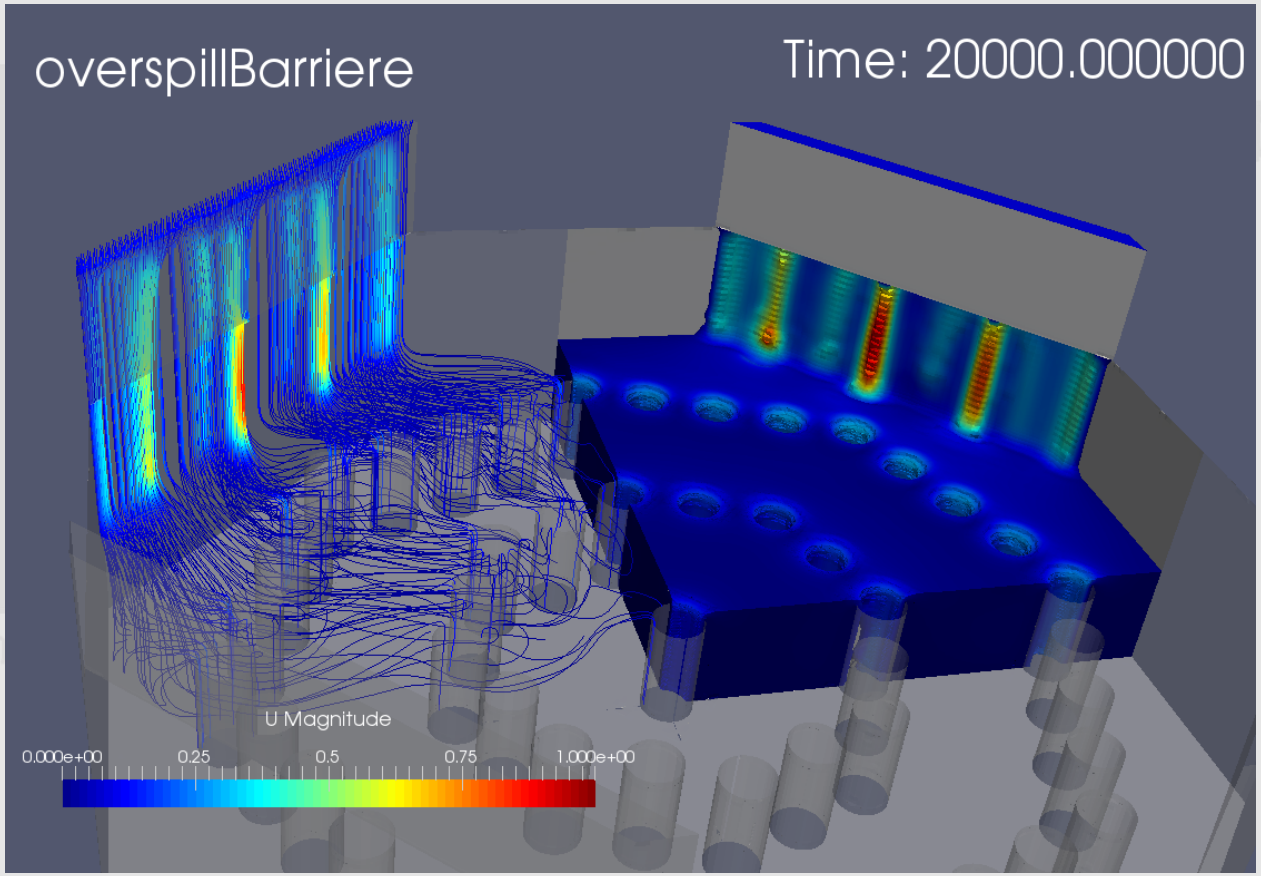
```
> qsub -p mpi 8 runUhdeUeberlauf.py templateCase test achteckBarrieren.parameters
```

- Runs on 8 processors of the cluster
- Everything works because the `pyFoamPrepareCase.py`-machinery is used

Outline

Outline

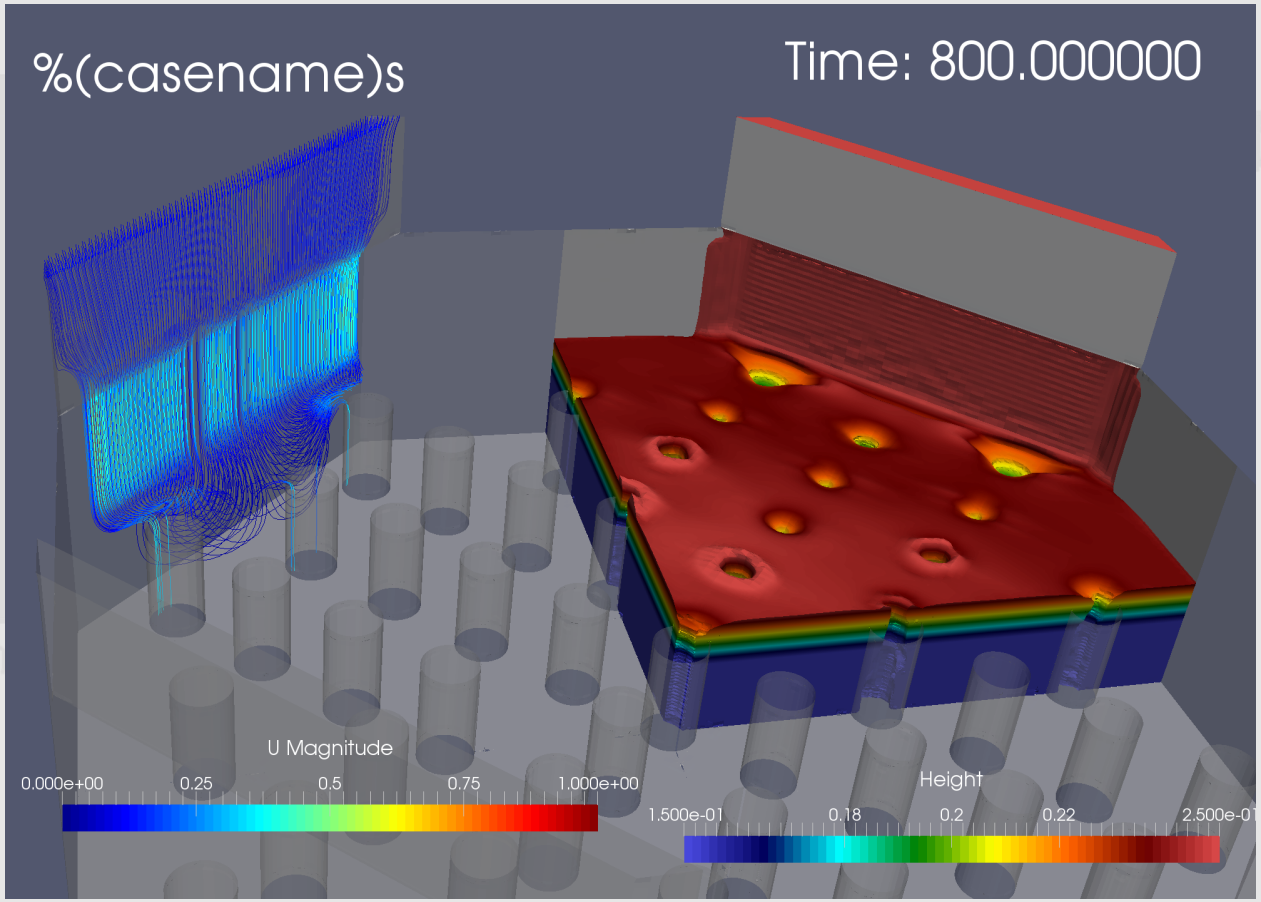
First result



Paraview state files

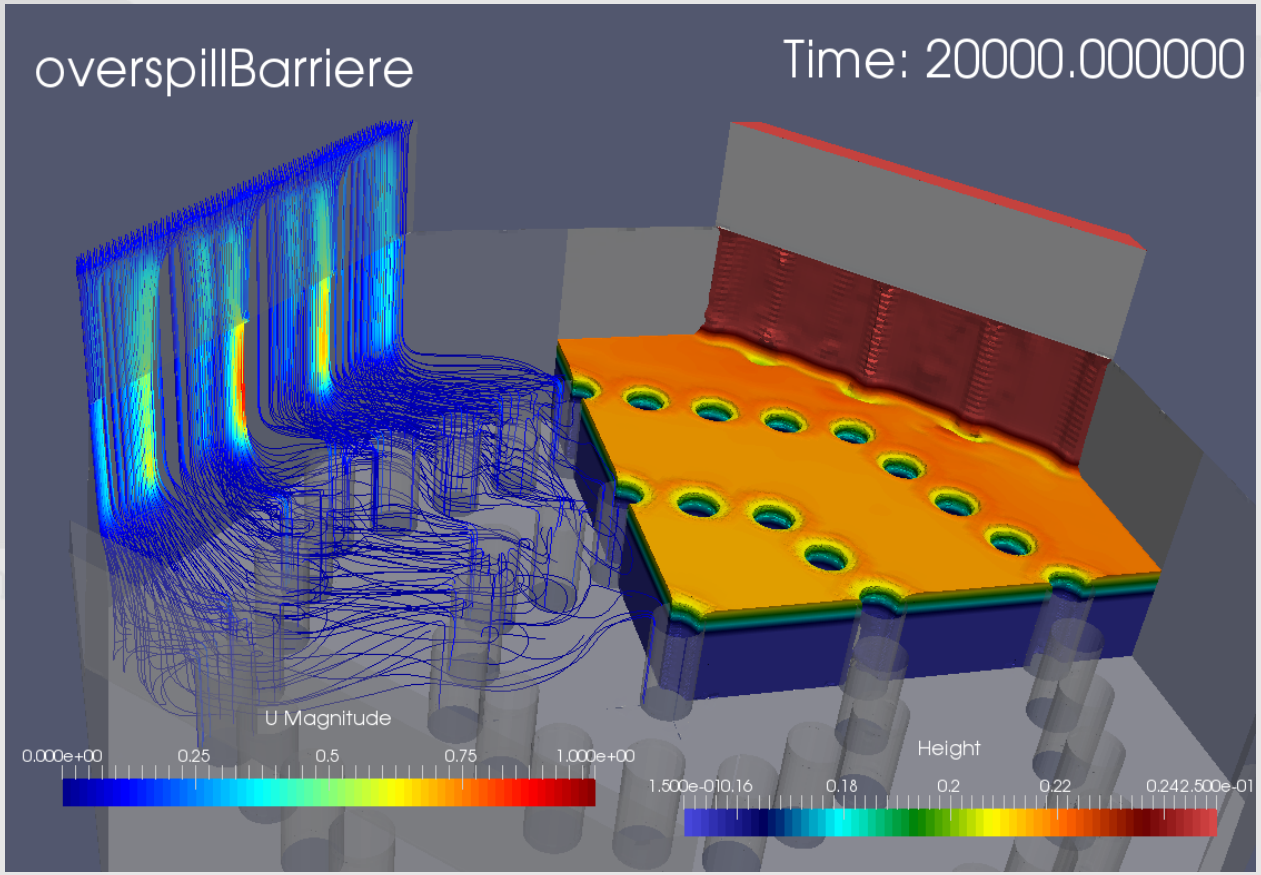
- Producing a plot like the previous one takes more than 10 minutes
  - If you exactly know what you're doing and what you want
  - and it is hard to get it to look exactely the same for a second time
- Everything that takes longer than 10 minutes should be scripted
- Solution: Paraview state files
  1. Set up the view the way you want it
  2. Save as a Paraview state file
  3. with the `pyFoamPVSnapshot.py` this state file can be applied to another case
     - `casename` is replaced with the actual case name in texts
     - other variables can be replaced as well
     - for some objects the colors can be replaced
     - timestep can be selected

Preparing the state file in Paraview



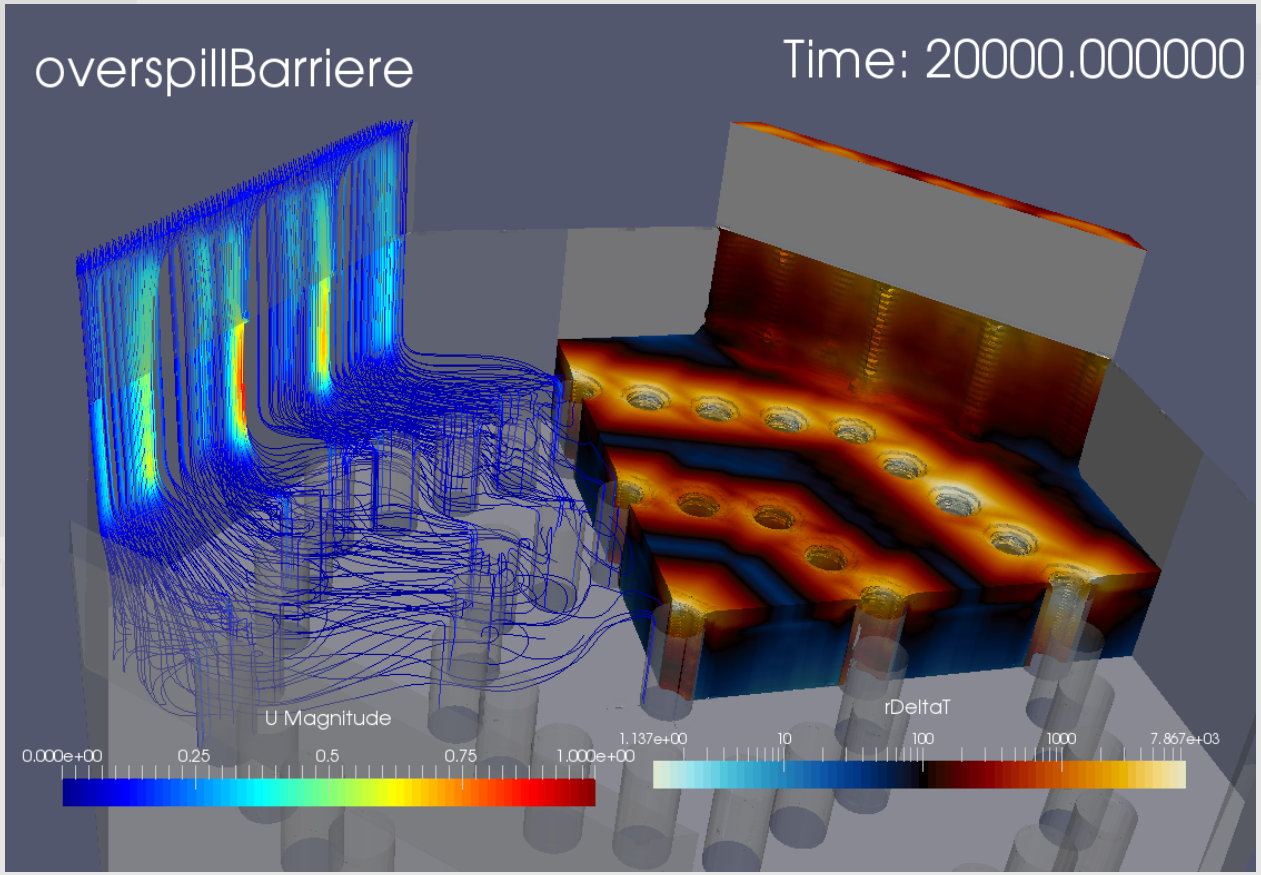
The actual height field

```
> pyFoamPVSnapshot.py --state=heightField.pvsm overspillBarriere --last-time
```



The timescale

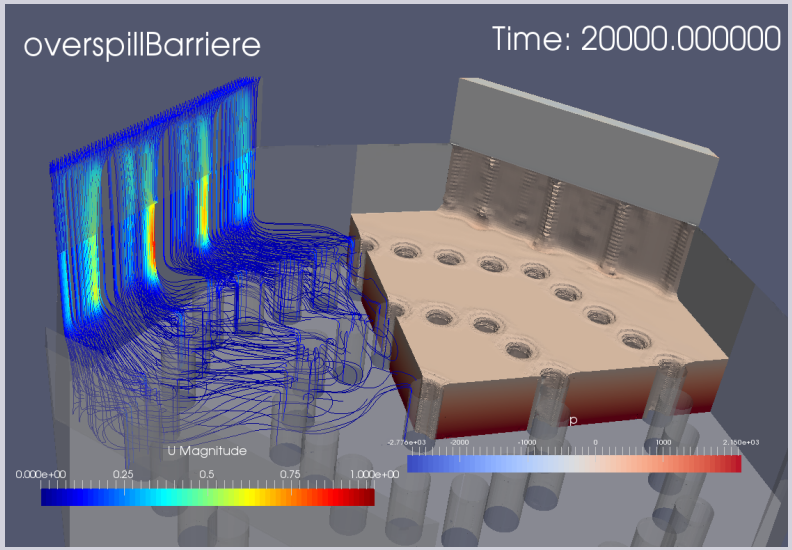
```
> pyFoamPVSnapshot.py --state=heightField.pvsm overspillBarriere --last-time --colors-for-<brk>
    <cont>filters="{'Calculator1':'rDeltaT'}"
```



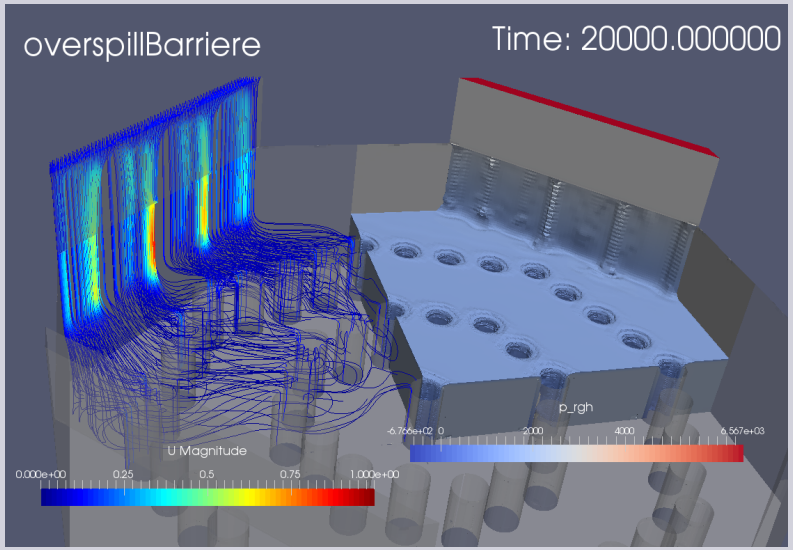
Two kinds of pressure

And two more pictures without using the mouse:

The "full" pressure



Minus gravity



Outline

Pictures are fine but numbers are better

- The pictures show the distribution of the flow
- But we want to know how much goes out of each outlet
- The function object `patchExpression` does calculations on patches
  - Patches are selected with the `patches` list
    - Regular expressions are possible as well
- All the accumulations from `swakExpression` are possible
- Output can be easily picked up by a `dynamic` PyFoam `customRegexp`

Calculating the flow

controlDict.template

- phi is OpenFOAM for "volume flow through a face
- the alias is necessary because fields in swak-expressions can't have . in their names

```
flows {
    type patchExpression;
    patches (
        "auslass.*"
        einlass
    );
    aliases {
        aSchmelze alpha.schmelze;
    }
    expression "|-densitySchmelze-|*phi*aSchmelze";
    verbose true;
    accumulations (
        sum
    );
    outputControlMode timeStep;
    outputInterval 1;
}
```

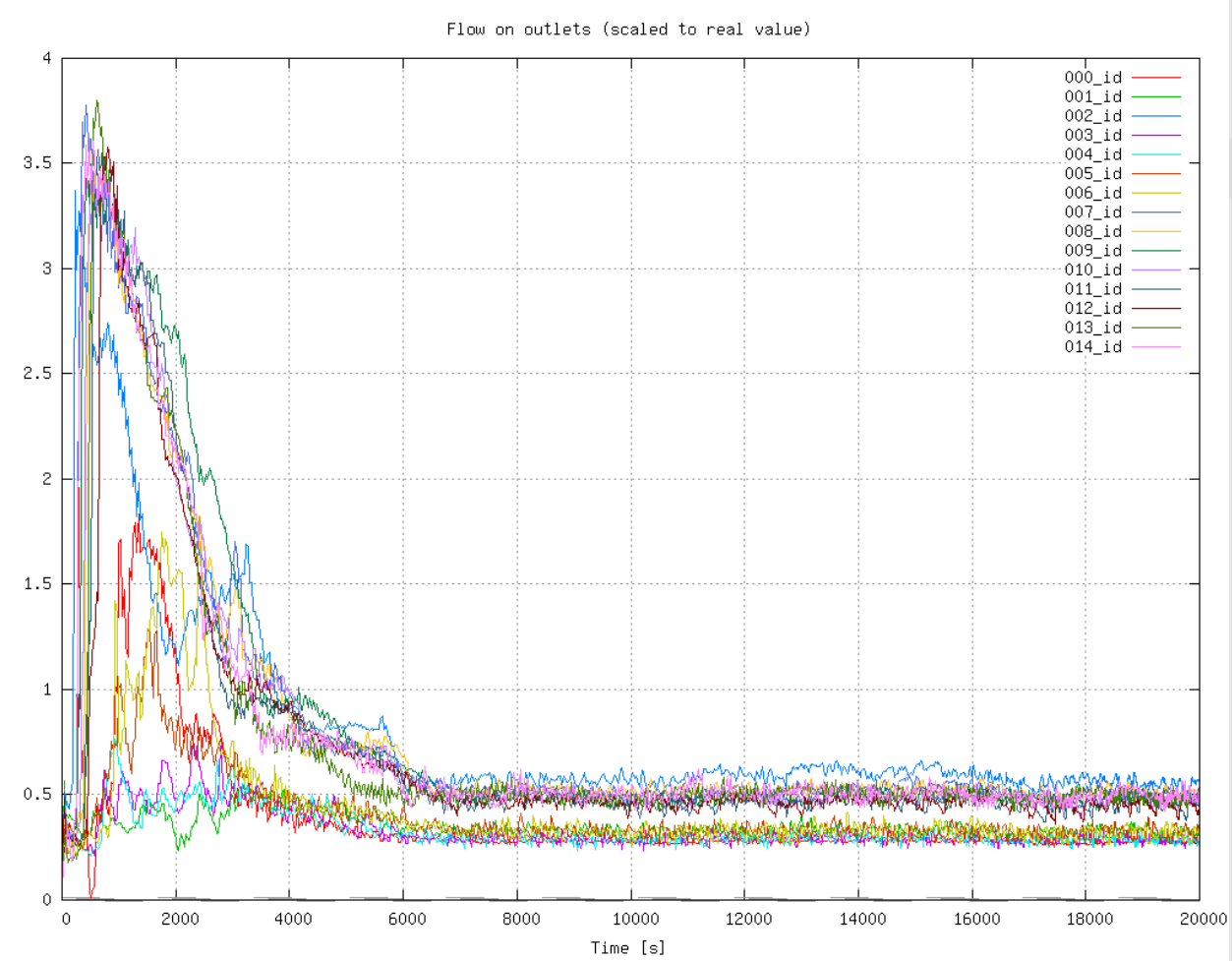
Correcting for "clipped" outlets

- Some outlets are not "complete"
  - The ones cut by the symmetry plane
- Flow on these has to be "normalized" with the known area

controlDict.template

```
<!--(if einbauRadius>0)-->
    flowsScaled {
        $flows;
        variables (
            // Theoretical area: |-einbauRadius*einbauRadius*3.1415-|
            "factor=|-einbauRadius*einbauRadius*3.1415-|/sum(area());"
        );
        expression "|-densitySchmelze-|*factor*phi*aSchmelze";
    }
<!--(end)-->
```

Mass flow on the outlets



Remote expressions in `swak`

- Usually in the `variables` list the expressions are calculated on the current patch (zone etc)
- Using a special syntax the values can be calculated on a different
  - Only condition: the expression must boil down to a single value
  - Syntax is `varname{patchname}`
- We use this to calculate the deficit of the flow
  - Sum of flows must be 0
    - Only then is the simulation converged

Calculating sum of flows and deficit

controlDict.template

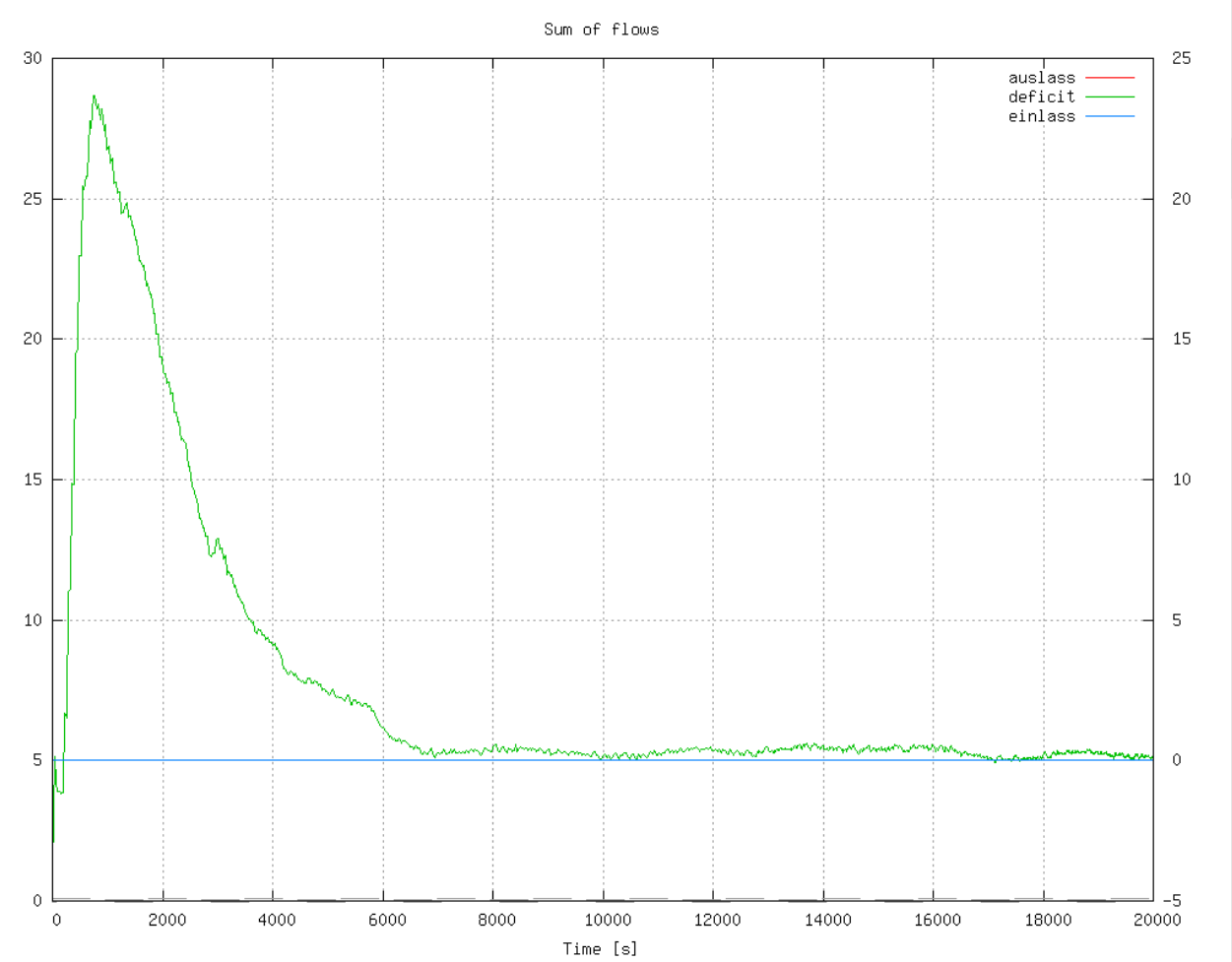
```
    flowSum {
        type swakExpression;
        valueType patch;
        patchName einlass;
        aliases {
            aSchmelze alpha.schmelze;
        }
        verbose true;
        accumulations (
            average
        );
        outputControlMode timeStep;
        outputInterval 1;
$$ auslaesse=["auslass%03d" % i for i in range(len(einbauSpec+einbauDickSpec))]
        variables (
<!--(for a in auslaesse)-->
            "val|-a-|{|-a-|}=sum(|-densitySchmelze-|*phi*aSchmelze);"
<!--(end)-->
            "valeinlass=sum(|-densitySchmelze-|*phi*aSchmelze);"
            "totalAuslass=|-'+'.join(['val'+an␣for␣an␣in␣auslaesse])-|;"
        );
        expression "totalAuslass";
    }
    flowDefizit {
        $flowSum;
        expression "totalAuslass+sum(|-densitySchmelze-|*phi*aSchmelze)";
    }
```

The actual calculation

controlDict

```
flowSum {
    type swakExpression;
    valueType patch;
    patchName einlass;
    aliases {
        aSchmelze alpha.schmelze;
    }
    verbose true;
    accumulations (
        average
    );
    outputControlMode timeStep;
    outputInterval 1;
    variables (
        "valauslass000{auslass000}=sum(1130.0*phi*aSchmelze);"
        "valauslass001{auslass001}=sum(1130.0*phi*aSchmelze);"
        "valauslass002{auslass002}=sum(1130.0*phi*aSchmelze);"
        "valauslass003{auslass003}=sum(1130.0*phi*aSchmelze);"
        "valauslass004{auslass004}=sum(1130.0*phi*aSchmelze);"
        "valauslass005{auslass005}=sum(1130.0*phi*aSchmelze);"
        "valauslass006{auslass006}=sum(1130.0*phi*aSchmelze);"
        "valauslass007{auslass007}=sum(1130.0*phi*aSchmelze);"
        "valauslass008{auslass008}=sum(1130.0*phi*aSchmelze);"
        "valauslass009{auslass009}=sum(1130.0*phi*aSchmelze);"
        "valauslass010{auslass010}=sum(1130.0*phi*aSchmelze);"
        "valauslass011{auslass011}=sum(1130.0*phi*aSchmelze);"
        "valauslass012{auslass012}=sum(1130.0*phi*aSchmelze);"
        "valauslass013{auslass013}=sum(1130.0*phi*aSchmelze);"
        "valauslass014{auslass014}=sum(1130.0*phi*aSchmelze);"
        "valeinlass=sum(1130.0*phi*aSchmelze);"
        "totalAuslass=valauslass000+valauslass001+valauslass002+valauslass003+valauslass004+valauslass005+<brk>
              <cont>valauslass006+valauslass007+valauslass008+valauslass009+valauslass010+valauslass011+valauslass012<brk>
              <cont>+valauslass013+valauslass014;"
    );
    expression "totalAuslass";
}
```

Sum and deficit



Actual distribution of the outflows

Using written data

- From the graph it is hard to tell which outlet receives how much fluid
- The numbers are in the files written by the function objects
  - But they don't tell us about the location
- Using a script the values of the flow are correlated with the positions of the outlets
  - Plotted as a "Bubble plot"
    - Area of the green circles corresponds to the mass-flow
    - Red circles are the positions

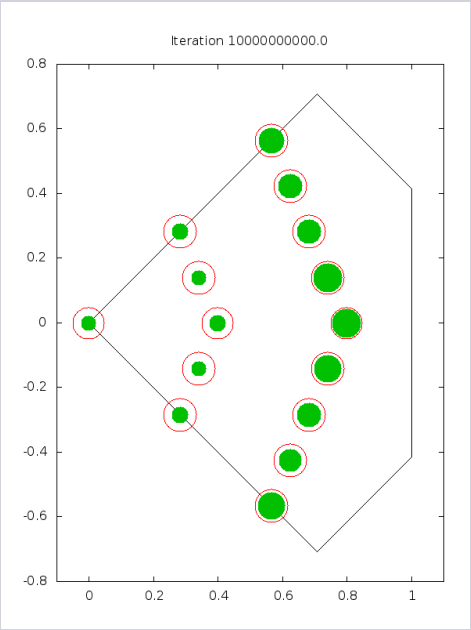
Bubble plot of the outflow

Actual distribution of the outflows

Using written data

- From the graph it is hard to tell which outlet receives how much fluid
- The numbers are in the files written by the function objects
  - But they don't tell us about the location
- Using a script the values of the flow are correlated with the positions of the outlets
  - Plotted as a "Bubble plot"
    - Area of the green circles corresponds to the mass-flow
    - Red circles are the positions

Bubble plot of the outflow

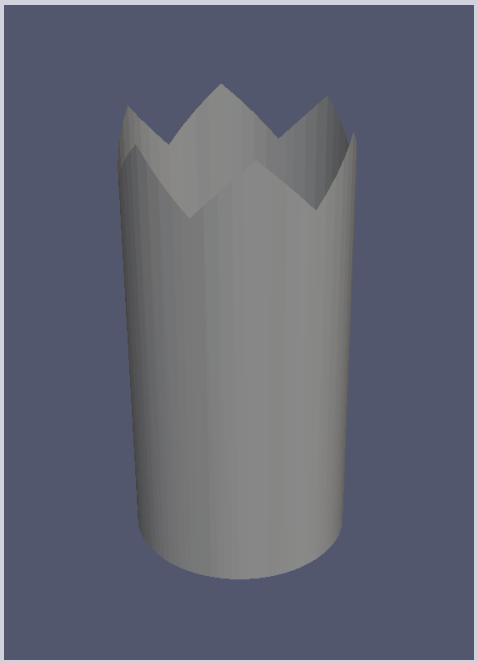


Outline

Outline

Outflow pipes with "crown"

The crown



Outlet geometry

- One strategy to achieve a more uniform distribution of the outflows is a different form of the pipes
- The "crowns" are supposed to make the flow more self-regulating

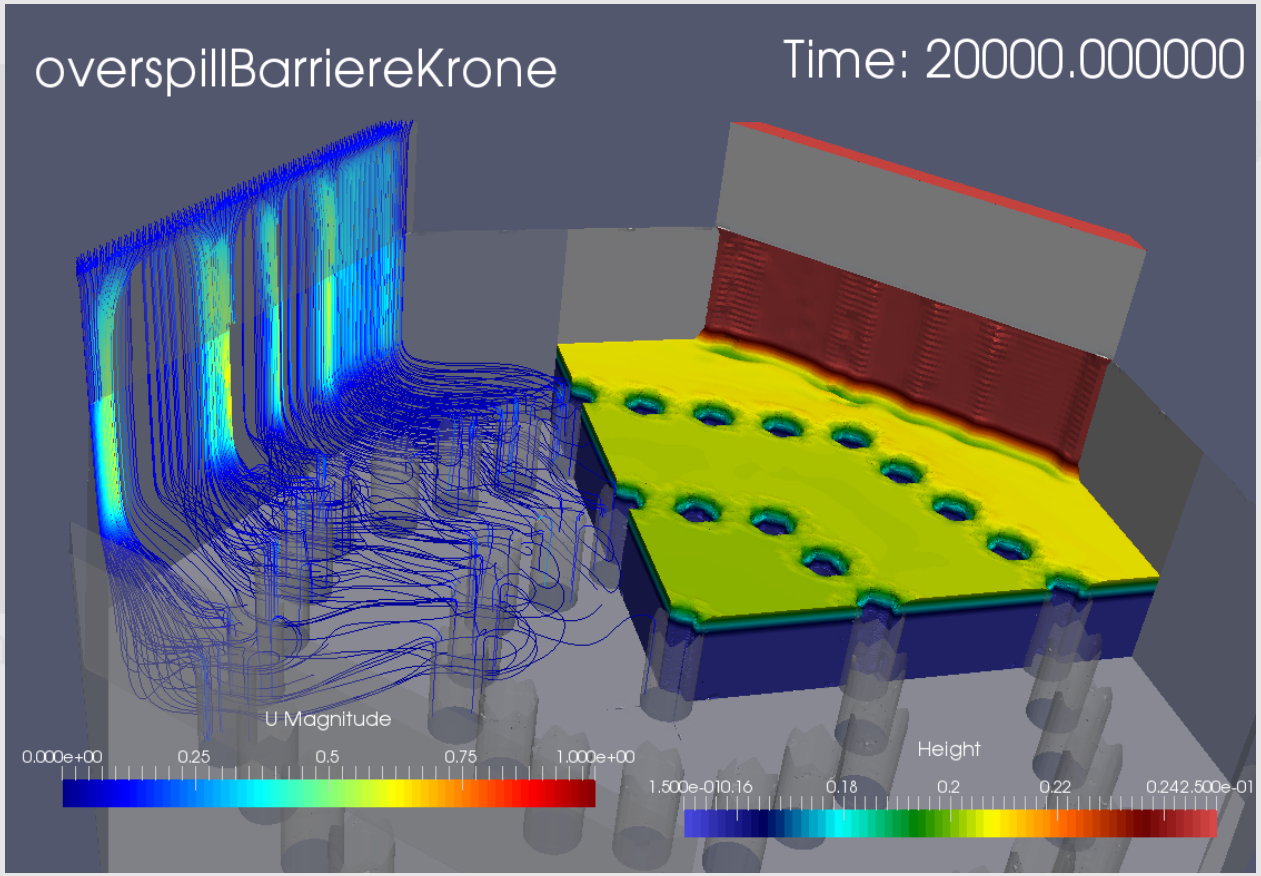
Setting the crown

- A new STL is prepared with the new form
- can be used with `defaultSTL`
  - that is used if the position specification has an entry `default`
- Just one line for a fundamental change in the case setup

achteckBarrierenKrone.parameters

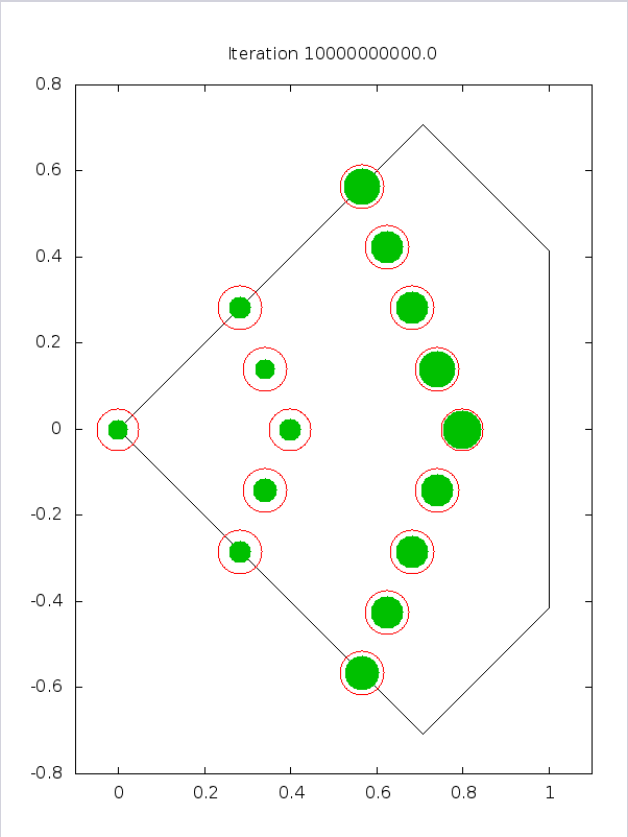
```
#include "achteckBarrieren.parameters"

defaultSTL Krone15;
```

Flow with crowns



Distribution of outflows with crown

Outflows



Interpretation

- The crowns don't seem to improve the situation significantly here
  - But the fluid level in the reactor is lower
- But other variations to the geometry might

Lowering the pipes

- Another possibility
  - Keep the design of the pipes
  - Just lower the inner pipes
    - That way they should get more flow
- In the specification we use the third coordinate to lower some pipes
  - The center pipe by $4cm$
  - The inner ring by $2cm$

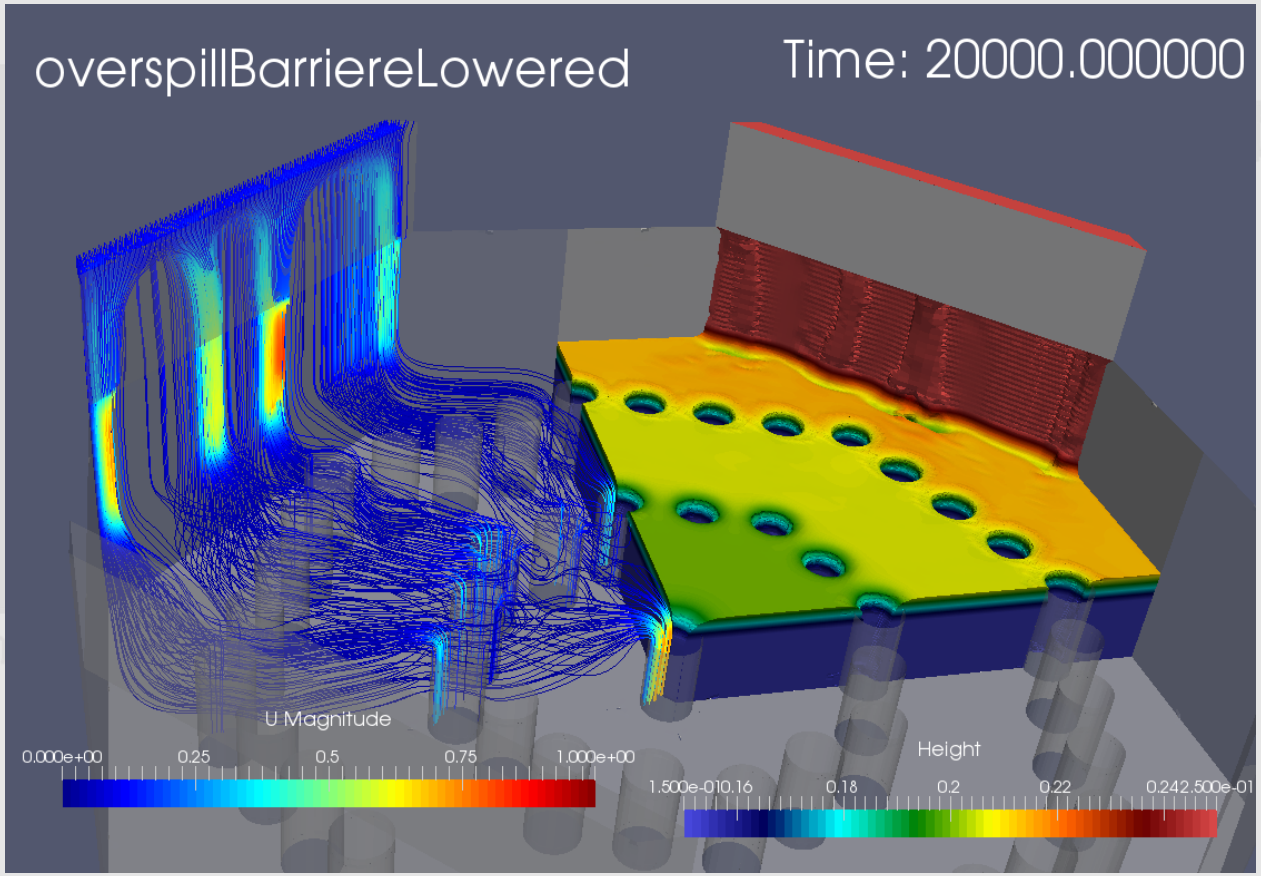
Variation with lowered pipes

- This needs a bit more editing
  - But still only one call to `pyFoamPrepareCase.py`

achteckBarrierenLowered.parameters

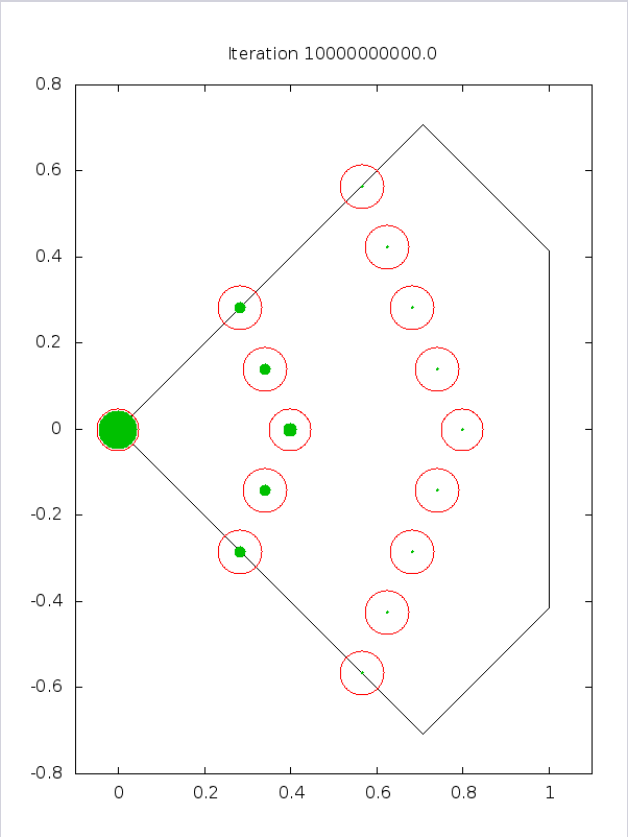
```
#include "achteckBase.parameters"

einbauSpec (
    (default    ( 0     0    -0.04) )
    (default    ( 2     0    -0.02) )
    (default    ( 4     0    0) )
    (default    ( 1     1    -0.02) )
    (default    ( 1    -1    -0.02) )
    (default    ( 0     2    -0.02) )
    (default    ( 0    -2    -0.02) )
    (default    ( 3     1    0) )
    (default    ( 3    -1    0) )
    (default    ( 2     2    0) )
    (default    ( 2    -2    0) )
    (default    ( 1     3    0) )
    (default    ( 1    -3    0) )
    (default    ( 0     4    0) )
    (default    ( 0    -4    0) )
);
```

Lowered pipes



Outflow analysis for lowered pipes

Outflow with lowered



Analysis

- Lowering definitely makes things worse
  - But we got that conclusion without actually building the reactor
  - Mixing this approach with the crowns nevertheless might be interesting . . .

Outline

Changing placements is easy

Placement strategies

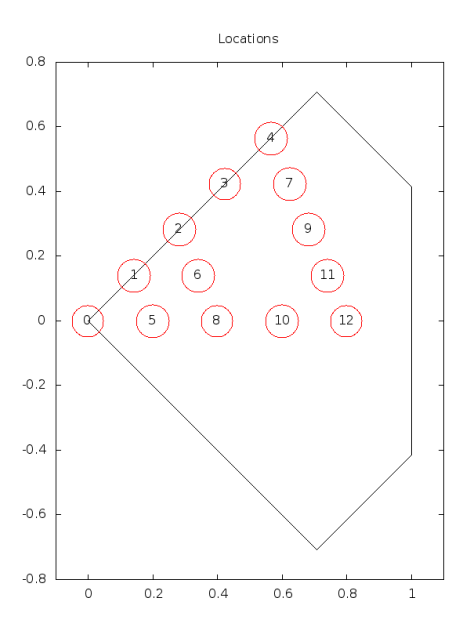
- Placing the pipes in a completely different way only needs editing one file
- We try 3 more strategies
  - Not all of them are good:
    - The flow has to be uniform to the downstream reactor. Not necessarily "per pipe"

Specifying the placement

```
#include "achteckBase.parameters"

einbauSpec (
    (default    ( 0     0    0) )
    (default    ( 0     1    0) )
    (default    ( 0     2    0) )
    (default    ( 0     3    0) )
    (default    ( 0     4    0) )
    (default    ( 1     0    0) )
    (default    ( 1     1    0) )
    (default    ( 1     3    0) )
    (default    ( 2     0    0) )
    (default    ( 2     2    0) )
    (default    ( 3     0    0) )
    (default    ( 3     1    0) )
    (default    ( 4     0    0) )
);
```

The three strategies

Asymmetry

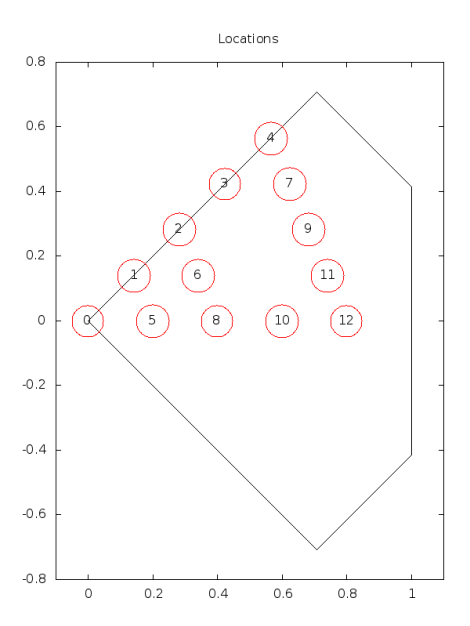


Radial

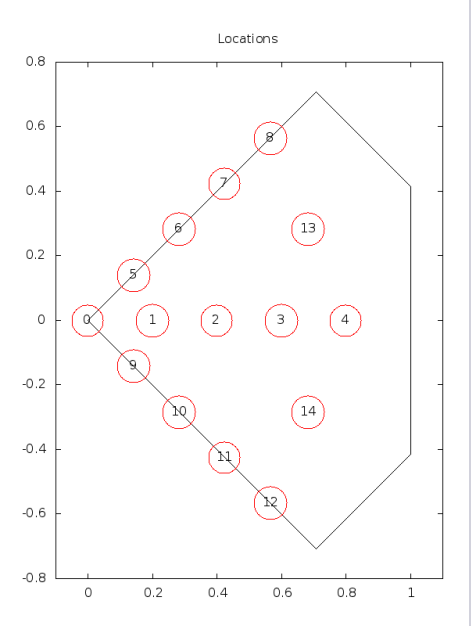
Lanes

The three strategies

Asymmetry



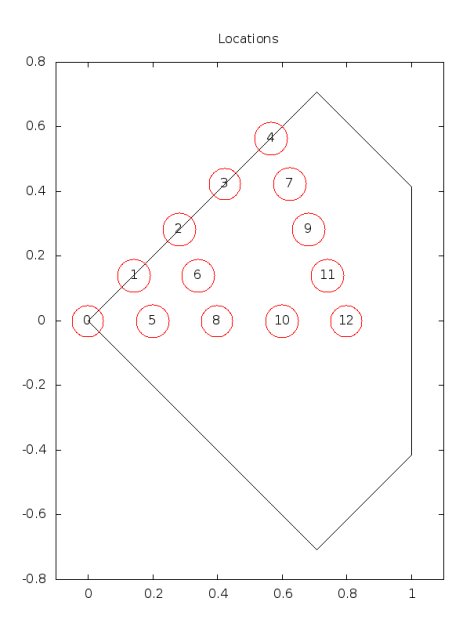
Radial



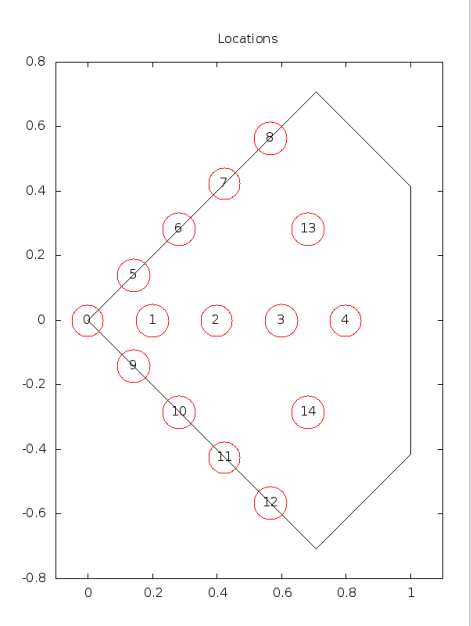
Lanes

The three strategies

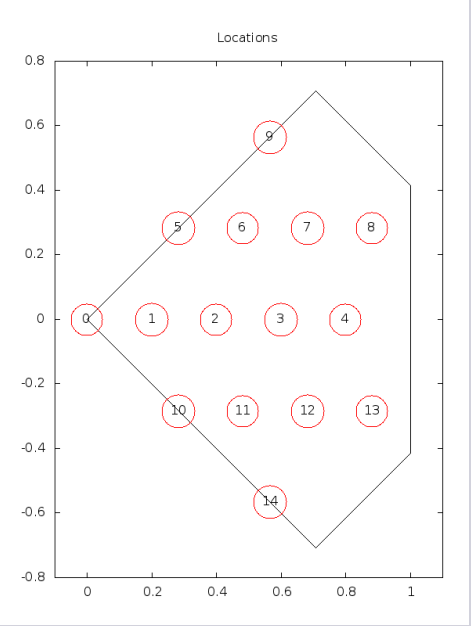
Asymmetry



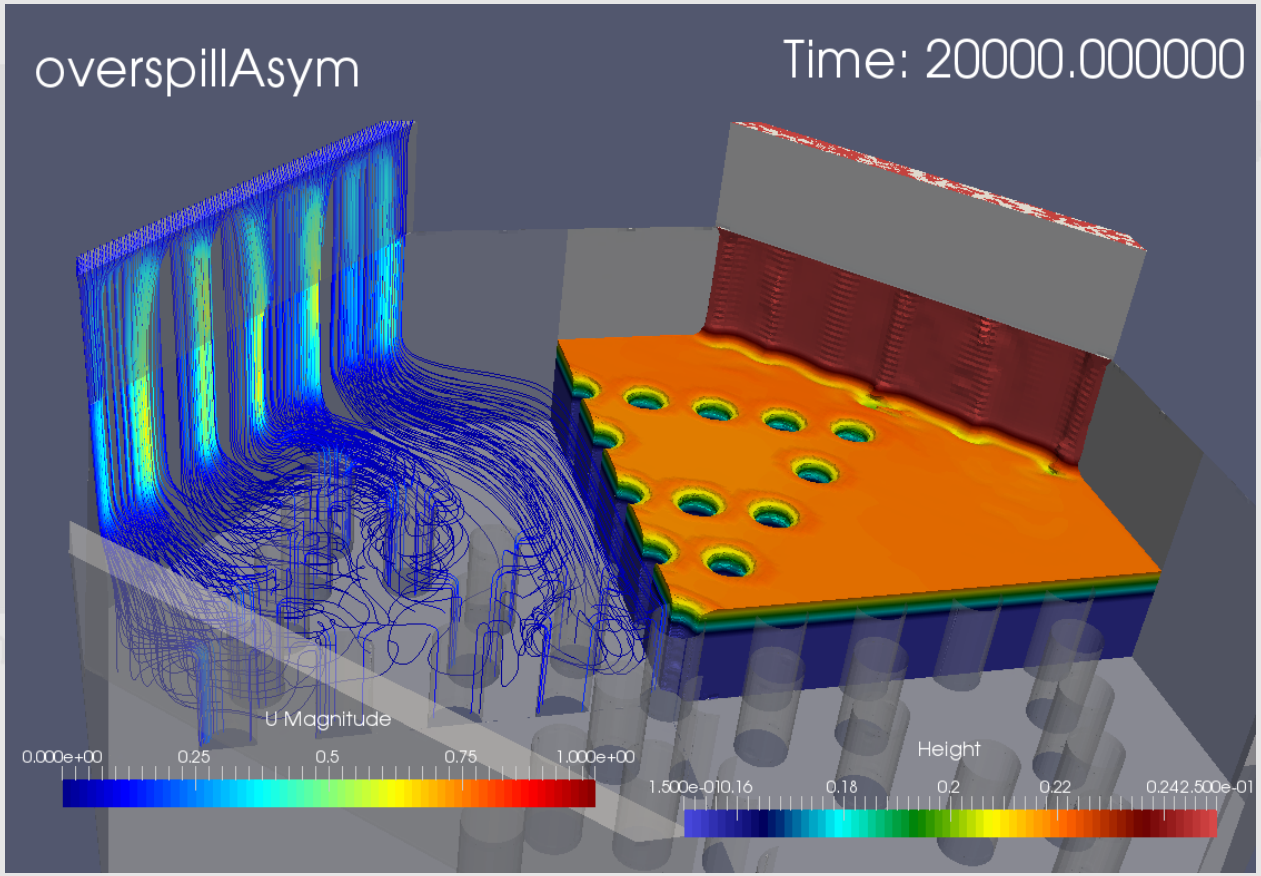
Radial



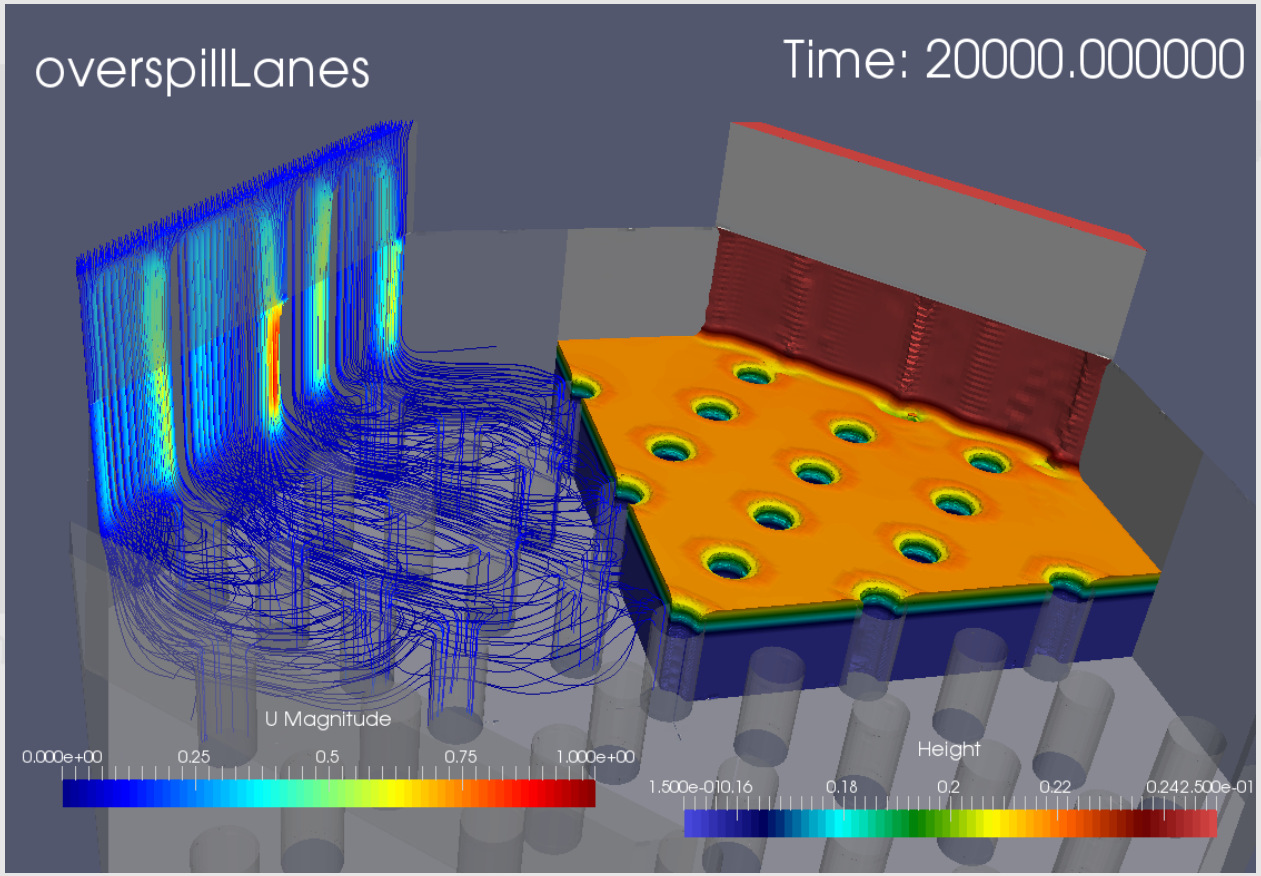
Lanes



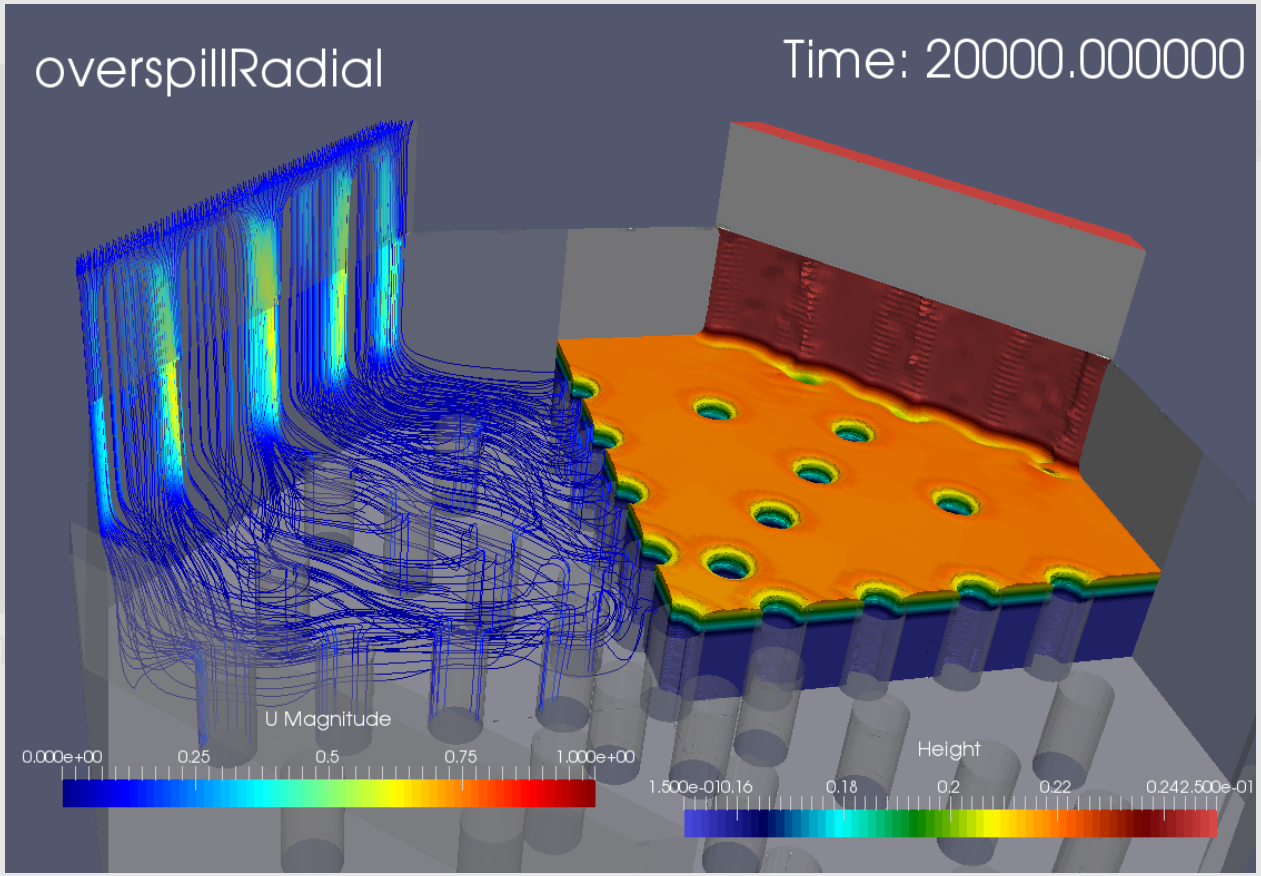
Asymmetric locations



Locations with lanes

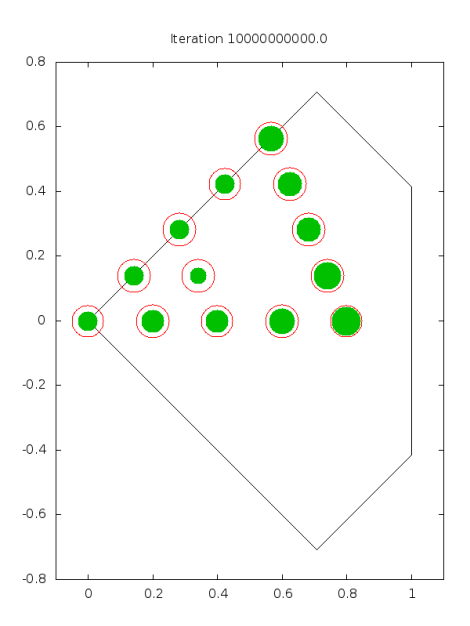


Radial placement

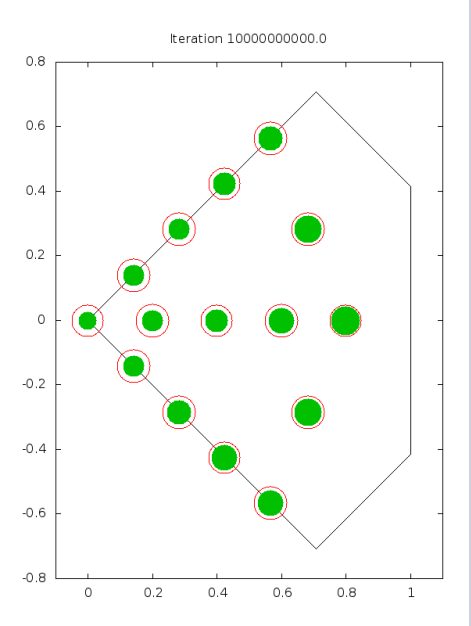


The outflow charts

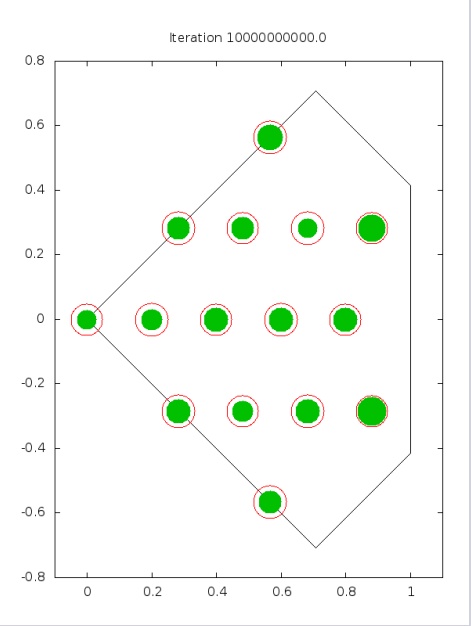
Asymmetry



Radial



Lanes



What is the best way to place the pipes?

- I don't know
- The reasons
  1. this is not the real geometry of the customer
  2. We set up the tools for the customer. They are now successfully doing the simulation themself
     - Non-CFD engineers
     - The template cases allow them to only change what needs changing
     - Quicker turn-around times because they can do case-setup and analysis in-house

Outline

Outline

Unmentioned features of swak

- Adding particles to solvers via function objects
- Custom fvOptions
- Better crash handling
- CGS operations on surfaces in snappyHexMesh
- Calculations on sets, zones, sampled sets, surfaces and particle clouds
- Integration of Python in function objects
- ...

Unmentioned features of PyFoam

- Packing and cloning cases
- Listing case directories
- Controlling OpenFOAM-runs over the network
- Quickly generating plots from timelines
- Manipulate `boundary` files
- Analyze logfiles after the simulation
- Use the library for your own scripts

Further information

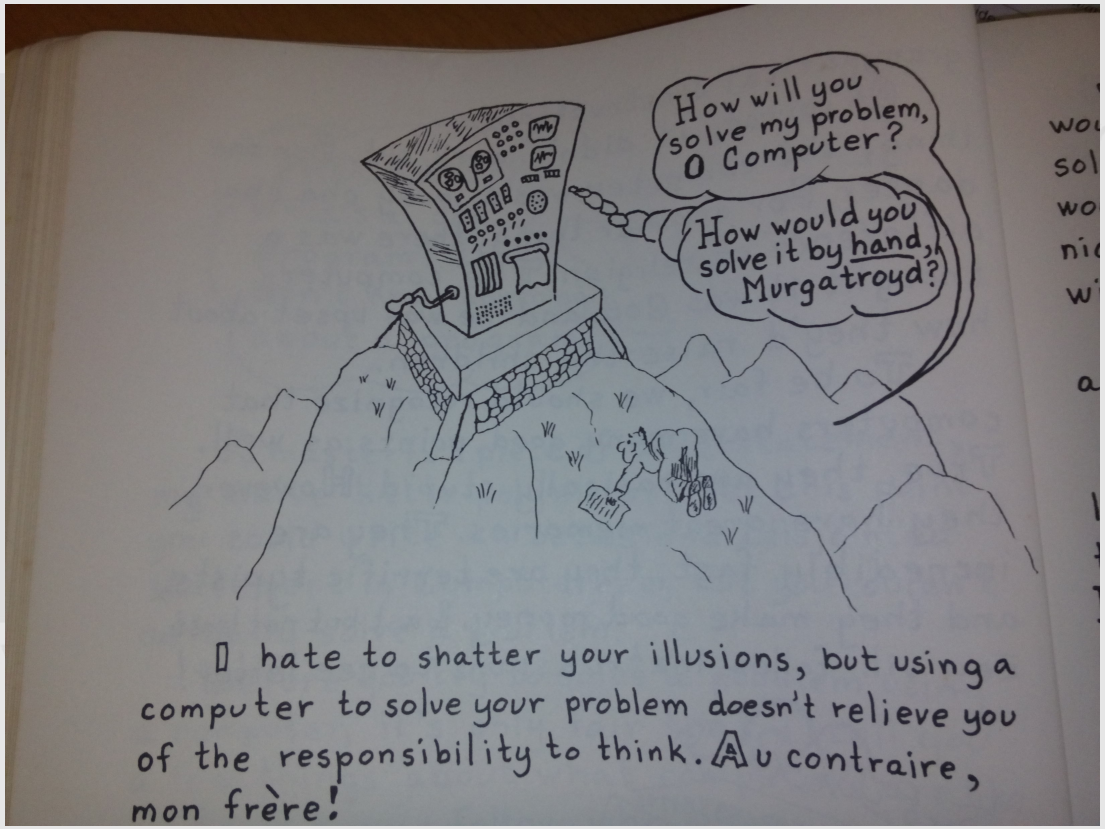
- Pages on the Wiki:

`https://openfoamwiki.net/index.php/Contrib/swak4Foam`
`https://openfoamwiki.net/index.php/Contrib/PyFoam`

- Twitter account announcing releases and new features: `@swakPyFoam`
- Presentations from OpenFOAM Workshops can be found at the two Wiki-pages above
  - Especially the basic training for swak and PyFoam allows doing everything by yourself
- There is a more complete presentation about `pyFoamPrepareCase.py` from the 10th Workshop

Outline

What these packages don't do for you



Picture taken from the "Fortran coloring book" by Roger Kaufman

Why use swak4Foam

- Because it helps to avoid the use of C++
  - CFD engineers shouldn't have to be programmers
  - Life is too short to program C++ all the time
- Reduces the number of "throwaway" C++ programs
  - Case setup and boundary conditions should be in the case. Not in a separate program or library

Why use PyFoam

- Does a lot of things for which usually throwaway `shell`, `sed` or `perl`-scripts are written
- One consistent set of tools
- With the `--help`-texts it is quite well documented
  - for the OpenFOAM-ecosystem

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start coding, tools or GUIs are in!

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start coding: svn or git in it

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text editor because GUIs are overrated
- It's a "program": use version control
  - Once you start coding: svn or git in it

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start coding: 100% of glitches in it

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start coding: don't or git in it

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start coding, you'll get in it

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start do `hg init` or `git init`

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start do `hg init` or `git init`

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start do `hg init` or `git init`

"Programming the case"

Using PyFoam and swak4Foam is a bit like programming

- ☹ Initial setup takes longer than usual
- ☺ Later simulations are quick to set up
- ☹ Things have to be tested
- ☺ It is harder to make mistakes afterwards
- ☹ There is nothing to click on
- ☺ Things are easy to automate
- ☺ You can use a text-editor because GUIs are overrated
- It's a "program": use version control
  - Once you start do `hg init` or `git init`

Outline

11th International OpenFOAM Workshop

- 26.-30. June 2016
- Takes place in Guimarães, Portugal
- Usual format
  - 2 days with presentations
  - 1 day with trainings
  - community with discussions, birds-of-a-feather sessions
- Further information at `http://www.openfoamworkshop.org`
- Looking forward to seeing you there
  - Pro tip: *Call for abstracts* starts at 1. January (to 18. March)

Thanks for your attention

Questions?

Thanks for your attention

Questions?