Expressive swak4Foam

Exploring the dark unknown corners

Bernhard F.W. Gschaider

HFD Research GesmbH

Duisburg, Germany
23. July 2019

Outline I

1. Introduction
   - This presentation
   - Who is this?
   - `swak4Foam`
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins

Outline II

- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Content

- This is about swak4Foam
  - The title of the presentation was a strong hint
- Different aspects of the expressions that are the heart of it
  - Various advanced topics that are rarely discussed
    - But allow pretty cool stuff
  - A bit of "theory" on the implementation
    - That explain some of the problems when using swak4Foam
- Not special function objects etc
  - Information here is applicable to almost any component of swak4Foam
- Some examples
  - But only sketches. The full examples can be found in the swak4Foam-sources

Intended audience

- People who have worked a bit with `swak4Foam`
  - or at least took the basic course in the previous session
    - basic stuff won't be spelled out
- Ever wondered why some expressions failed with a strange error?
  - this presentation is for you
- Ever thought "there must be a way to do this"?
  - this presentation is for you

Format

- Different topics will be covered
  - Only small examples
- Therefore this training will be purely "lecture style"
  - No exercises on the computer
- If names of example cases are give they can be found in the `Examples` directory of the sources
  - For instance `groovyBC/pulsedPitzDaily` is found at `Examples/groovyBC/pulsedPitzDaily`

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Bernhard Gschaider

- Working with OpenFOAM™ since it was released
  - Still have to look up things in `Doxygen`
- I am not a core developer
  - But I don't consider myself to be an *Enthusiast*
- My involvement in the OpenFOAM™-community
  - Janitor of the `openfoamwiki.net`
  - Author of two additions for OpenFOAM™

    swak4foam Toolbox to avoid the need for C++-programming

    PyFoam Python-library to manipulate OpenFOAM™ cases and assist in executing them
  - In the admin-team of `foam-extend`
  - Organizing committee for the *OpenFOAM™ Workshop*
- The community-activies are not my main work but *collateral damage* from my real work at . . .

Heinemann Fluid Dynamics Research GmbH

The company

- Subsidary company of *Heinemann Oil*
  - Reservoir Engineering
  - Reservoir management

Description

- Located in Leoben and Vienna, Austria
- Works on
  - Fluid simulations
    - OpenFOAM™ and Closed Source
  - Software development for CFD
    - mainly OpenFOAM™
- Industries we worked for
  - Automotive
  - Processing
  - ...

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

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

Supported versions and release policy

- swak4Foam has no fixed release schedule. Releases happen
  - when I have time for it
    - try to do it 2 times a year
  - when there were releases of OpenFOAM-forks recently
    - and the code is sufficiently stable
    - tested against all supported forks
- The supported OpenFOAM-versions (in the forthcoming release) are
  - latest released ESI OpenFOAM-release (currently v1906)
  - latest version of the nextRelease-branch of foam-extend
  - latest released Foundation OpenFOAM-release (currently 7)
  - OpenFOAM 2.3
    - this makes sure that OpenFOAM-releases between that are not broken
- If your OpenFOAM-release is not yet supported look at the develop-branch of the source repository

Available documentation

There is not much documentation on swak4Foam

- Documentation that comes with sources of swak4Foam (and is maintained in parallel with it)
  1. the README-file
  2. the *incomplete* reference manual
  3. the example compatibility matrix
- Various presentations
  1. here at the workshop
     - training on specific topics
  2. presentations at PFAU (Austrian User Group meeting)
     - mostly descriptions of new features

They can be found on the Wiki https://openfoamwiki.net

README

Contains

- how to build swak4Foam
- how to develop for swak4Foam
- short descriptions of the things in the package
- News in the releases
  - every time a feature is added this is updated
    - so even the develop-branch always lists changes to the last release
    - usually a description of the feature is given here

Reference manual

- This is called the Incomplete *Reference Manual*
- It is a work in progress
  - Existing parts are adapted if anything changes
  - New parts are written when I find time
    - I hardly have time
- Completed parts are
  - the parser and expressions
    - all built-in functions and operators
    - structure of dictionaries (`variables` etc)
  - bits and pieces
    - scripting language integration
    - state machines
- Missing parts are
  - List of utilities
  - List of function objects
  - List of boundary conditions

Compatibility matrix

- There are two orthogonal factors
  1. The 4 supported versions
     - Which are sometimes incompatible in their dictionary format
  2. Lots of cases in the `Examples`-directory
- It is hard to tell if all $4 \times N$ combinations work
  - Some cases have features that are not supported by all forks
- The *Example Compatibility Matrix* tries to keep track of these combinations
  - Only for a sub-set of the existing examples
    - But new examples are usually added to the matrix
  - It is recorded here which combinations work
- To make sure that cases work with different versions `pyFoamPrepareCase.py` is used
  - There exists different presentation on that
    - But usually this command is sufficient to prepare cases

```
pyFoamPrepareCase.py .
```

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Expressions in swak4Foam

- The evaluation of expressions is the core functionality of `swak4Foam`
- In the dictionaries the expression manifests as a simple string
  - between " and "
- The string is read and transformed into an evaluation
  - The result of the evaluation is used
    - To modify fields
    - Print information
    - other stuff
- The content of the expression string has to follow some rules
  - The grammar
    - If the string doesn't conform to the grammar it is *syntactically incorrect*
  - The act of checking that the expression conforms to the grammar is called *parsing*
  - The grammar has to "make sense"
    - For instance: you can't add a scalar to a vector
- When *parsing* is finished `swak4Foam` konws "what to do"

What is a parser?

- In swak4Foam a parser is a grammar and the part of the program that supports it
- The design principles for the swak4Foam-parsers are
  1. Syntax should be similar to OpenFOAM C++-expressions
     - Not always possible
  2. "Least surprise": only obvious actions should be done
     - swak4Foam should not "guess" what the user wants
     - things have to be written down explicitely
  3. Backward-compatibility
     - grammar should not change once it is released
     - "write once: use for centuries"

Parser / Lexer /Driver

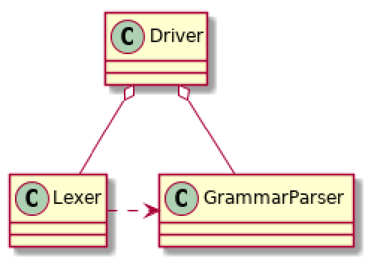
The trinity

lexer breaks the string into *tokens* like `3.1415`, `+`, `rho`

parser takes the tokens and checks that they conform to some rules like "scalar + scalar is a scalar"

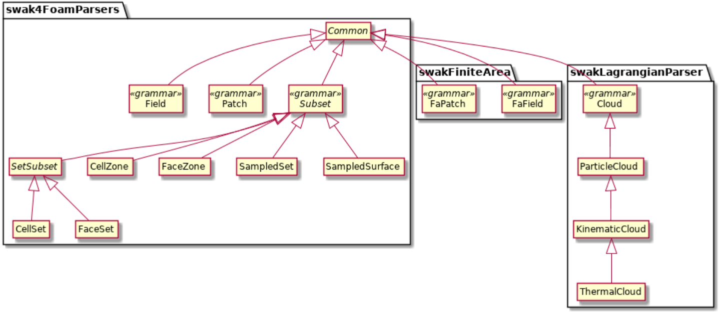
driver the part that does the actual calculations (add the two scalars and store the results in the correct location)

Relationship Driver/Lexer/Parser



The parsers

- Here are the different libraries the parsers reside in
    - With the parsers in them
- The ones with « grammar » are actual grammars
    - The others are "just" different drivers



Selection of the correct parser

There are two ways to get a parser in `swak4Foam`

1. You can't select it because it is hardcoded (because there is only one "natural" parser)
   - for `groovyBC` only calculations on a patch make sense
   - full fields are calculated with `funkySetFields`
2. the wanted parser has to be selected
   - for instance in the `swakExpression` function object
   - usually via a `valueType` directory entry
     - some parsers need additional information (patch name for instance)

Like everything in OpenFOAM parsers are run-time selectable

- the *banana* trick applies
- additional parser can be added via libraries

Parser names

These are the currently existing parsers selectable via `valueType`

name	Description
`internalField`	Calculation on the internal values of a field
`patch`	Calculation on a boundary patch
`faceZone`	On a `faceZone` of the mesh
`faceSet`	On a `faceSet`
`cellZone`	Calculation on a `cellZone`
`cellSet`	Set of cells
`set`	Calculation on a `sampledSet`
`surface`	Calculation on a `sampledSurface`
`cloud`	Calculation on a cloud of lagrangian particles
`internalFaField`	Internal values of a FAM-field (not all forks)
`faPatch`	Boundary patch of a FAM-field (not all forks)

Parser looks for a name

When the lexer encounters a field name `T` it

1. Looks for something called `T`
   - See two next slides how that works
2. Inspects it
   - Looks for the value type: `scalar`, `tensor`, . . .
   - Looks for the field type: volume field, surface field . . .
     - not always necessary
3. Reports back to the parser
   - "token `T` is a scalar volume field"
4. The parser tries to make sense of it
   - "I can add that to the scalar I got before"
     - Goes on parsing

   or
   - "can't add this scalar volume field to the vector surface field"
     - Fails with a syntax error

How the parser finds fields

Two options

1. File of that name on disk
   - This only happens for pre/post-processing utilities like `funkySetFields`
   - function objects and boundary conditions don't do this
     - would be a performance desaster
     - . . . and inconsistent
2. Objects in memory
   - OpenFOAM has a data structure called "the `objectRegistry`"
     - almost all fields are registered there
     - OpenFOAM uses it for things like automatically writing files
     - it is brilliant: *swak4Foam wouldn't work without it*
   - when `swak4Foam` encounters a name it looks for it there
     - if found it checks whether it matches a required type

Name resolution order

swak looks for names in that order (first match wins)

1. name of a mesh (not discussed here)
2. a `timeline` (not discussed here)
3. a `lookuptable` (not discussed here)
4. a 2D lookup table (also not discussed. See *Incomplete reference*)
5. Variable
   - possibly shadowing a field of the same name
     - there is a warning for that
6. Field
   - on disk or in memory
   - possibly using `aliases` (see below)
7. Plugin functions (see below

Common options

When initializing a parser `swak4Foam` looks for some optional parameters in the same sub-dictionary

variables
: most often used: variables to make things more readable

storedVariables, delayedVariables
: we'll talk about these later

timelines, lookuptables, lookuptables2D
: getting functions from datafiles. Look at the reference manual for details

searchOnDisc, searchInMemory, cacheReadFields
: Change the way `swak4Foam` looks for fields. Hardly needed

debugCommonDriver, traceParsing, traceScanning
: Switches for debugging the parser. Usually only needed by developers

aliases
: we'll get to that later

Make sure that it fails

Not really a swak4Foam topic

- OpenFOAM tries to be tolerant about configuration errors in function object
  - if there is a configuration error then OpenFOAM just prints a warning
  - happily goes on simulating
  - ... but without the results of the function object
- Personal opinion
  - This is not a good idea
    - Because I run OpenFOAM for the results. Not to burn CPU-hours
- As some swak4Foam-setups are non-trivial there is a good chance that you make mistakes the first time around
  - But OpenFOAM makes you believe that all is good
- To get the old behaviour set this environment variable

```
export FOAM_ABORT=1
```

- that way OpenFOAM fails for every swak4Foam-problem

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Everything happens on the mesh

- Although the expressions look continuous they are evaluated on discrete elements
  - cells, points, faces
- Not every function is defined on cells/points/faces
  - Sometimes a operation changes the mesh element type of the result
- Sometimes the straightforward implementation gives wrong results because of these differences
  - Example on the next slide
- `swak4Foam` offers ways to convert between mesh element types

Points vs faces

Consider these implementations of a parabolic inlet condition

Get range from face centers

This is how people usually do it the first time

```
movingWall
{
    type            groovyBC;
    value           uniform (1 0 0);
    variables (
        "xMin=min(pos().x);"
        "xMax=max(pos().x);"
        "x=pos().x;"
    );
    valueExpression "normal()*(x-xMin)*(<brk>
        <cont>xMax-x)";
}
```

pos() is the positions on the face centers

Get range from points of the mesh

This is how it should be done

```
movingWall
{
    type            groovyBC;
    value           uniform (1 0 0);
    variables (
        "xMin=min(pts().x);"
        "xMax=max(pts().x);"
        "x=pos().x;"
    );
    valueExpression "normal()*(x-xMin)*(<brk>
        <cont>xMax-x)";
}
```

pts() is the positions of the mesh points

Comparing implementations

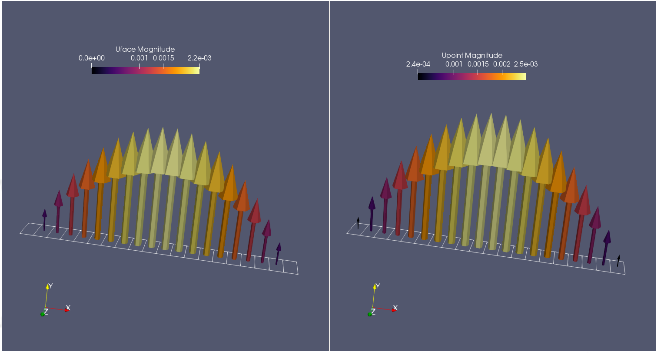


Figure: "Boundary cells" are zero and maximum is wrong on the left side

The left implementation has a wrong mass-flow

Native and secondary mesh elements

- Every parser has a preferred mesh element it works on
  - We call this the native structure
  - This is usually the "natural" element for the FVM
    - For instance: for patches the *native* structure is the `face`
    - For fields the *native* structure is the `cell`
- There is another element type that is the build block of the native structure
  - We call this the secondary structure
    - For instance: for patches the *secondary* structure is the `point`
  - Fields are special because they have two *secondary* structures: `face` and `point`

Native and secondary for the different parsers

Parser	*native* structure	secondary structure
`internalField`	Cell values	Face values and point values
`patch`	Face values	Point values
`faceZone`	Face values	*none*
`cellZone`	Cell values	*none*
`faceSet`	Face values	*none*
`cellSet`	Cell values	*none*
`set`	Values on sample points	*none*
`surface`	Values on the facets	vertices - not yet implemented
`cloud`	Values on the particles	*none*
`internalFaField`	Area (face) values	Edge values
`faPatch`	Edge values	Point values

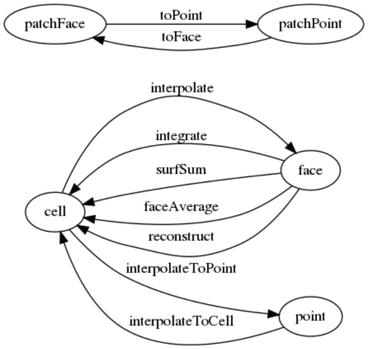
Interpolating values

Going to other structures

- To go from one structure to another there are interpolation functions
- Based on regular OpenFOAM-functionality
- Caution: interpolating usuallyresults in information loss
- Interpolate temperature field to faces:

```
interpolate(T)
```

The functions



Constants are always "native"

- One problem that people usually have is that constants are "only" native
  - This comes from the "least surprise"-principle
- To use on secondary structure the constant has to be interpolated

doesn't work

```
mag(pts()-vector(0,1,0))
```

works

```
mag(pts()-interpolateToPoint(vector(0,1,0)))
```

The weight

- most parsers have a function `weight()`
  - this is the property that would normally be used as a weighting factor when averaging
    - Cell volume `vol()` for the `internalField`
    - Face area `area()` for everything "flat": patches, sampled surfaces, ..
    - 1 for sampled sets
    - For "simple" particles it is 1
    - For `KinematicCloud` and up it is the *parcel mass*: particle number times the particle mass
- allows general writing of expressions
  - they can be re-used on cells and faces without rewriting
    - `sum(T*weight())/sum(weight())` instead of
    - `sum(T*vol())/sum(vol())` on the =internalField
    - `sum(T*area())/sum(area())` on a patch
- this is the property that is usually used for the accumulation `weightedAverage`
  - and all other accumulations with `weighted` in the name

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

What are uniform expressions

- OpenFOAM has two ways to store `Fields`
  - uniform the whole field has the same value
  - nonunifom at least one element is different from all the others
- But in memory `uniform` needs the same amount of storage
- When `swak4Foam` stores intermediate results it makes a similar distinction
  - if all values are the same the value is marked as `uniform`
    - needs less memory as well
- When assigning to a OpenFOAM-structure the whole structure is set to the same value
- Uniform values can be used interchangably
  - No interpolation needed
- Functions that generate uniform values:
  - `min` / `max`
  - `average` / `sum`
  - `minPosition` / `maxPosition`

Where do I need uniform expressions

- Sometimes uniform values are required
- Mostly in situation interpolating to different entities would be required
    - For instance from one patch to another patch
    - The reason is:
        - "least surprise"

Accumulations

Related but not the same:

- When printing expression results the full field would be too long
  - Should be broken down to a single value
  - This is done by the `accumulations`
  - Some accumulatons are
    - min / max
    - average / weightedAverage
    - sum /weightedSum
    - . . . .
- But this is only a post-processing thing
  - Not stored

Outline

Reasons why fields are not found

Often we get an error message

```
field foo not existing or of wrong type
```

Possible reasons are

- You mis-typed the name and it is really not there
  - Sometimes the computer is right (rarely)
- The field is of the wrong type
  - `swak4Foam` tells you what type it expects
    - For `p+U` it probably will complain because it expected a `volScalarField`
- There was a field of that name. But not anymore
  - For instance the thermophysical libraries like to create temporary fields vor c_p etc
    - They are removed when they are not needed anymore
  - The field exists but the `objectRegistry` doesn't know it
    - That can happen if a second temporary field with the same name is created. It "kicks" the first one out of the registry
- The field doesn't have a name `swak4Foam` can handle

Getting a list of things

- swak4Foam has a function object that lists all the fields that currently in memory

functions in system/controlDict

```
whatIsThere {
    type listRegisteredObjects;
}
```

Output

```
Content of object registry region0
                    Name                          Type Autowrite
======================== ============================= =========
                       K                      IOobject No
                     K_0                      IOobject No
           MRFProperties                      IOobject No
                       T                volScalarField Yes
<snip>
            thermo:alpha                      IOobject No
               thermo:mu                      IOobject No
              thermo:psi                      IOobject No
              thermo:rho                      IOobject No
thermophysicalProperties                    dictionary No
    turbulenceProperties                    dictionary No
```

Valid names in OpenFOAM vs swak4Foam

- What `swak4Foam` considers a valid field name is a sub-set of the valid field names in OpenFOAM
  - only letters, digits and _
    - may not start with a digit
- OpenFOAM is more liberal
  - Characters like `:` `.` are allowed in a name
    - These characters have different meanings for the `swak4Foam`-parsers
  - Everything that is a `word` for OpenFOAM can be a field name
- This means that fields like `thermo:rho` or `alpha.water` are not accessible for swak-expressions
  - But we want to access them

Aliases

- Workaround is a lookup table that says "if you see this *swak* name we really mean this *OpenFOAM* name"
  - This is a dictionary called `aliases` in the dictionary that has the parameters for the parser

aliases

```
aliases {
    rhoAir thermo:rho;
    alphaWater alpha.water;
    gasPressureNameForPeopleWhoLikeLongNames p;
}

expression "(1-alphaWater)*rhoAir";
```

- Try to make sure that your aliases don't have the same name as existing fields
  - This can lead to weird results
- Of course you can have aliases for field names without special characters

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

"Don't repeat yourself"

- Configuration dictionaries for `swak4Foam`-components can be quite long
  - `expression`, `valueType`, `variables` . . .
  - Sometimes we need many similar evaluations
    - then changes need to be done in many places
  - OpenFOAM helps with that
    - the mechanism is called *macro expansion*
- `swak4Foam` evaluations are usually related to the OpenFOAM-simulation
  - Sometimes a constant from the OpenFOAM-configuration is needed in the swak-expression as well
  - if it has been "copied" over then it has to be changed once the original is change
    - otherwise the evaluation is wrong
  - `swak4Foam` has a mechanism to help with that
    - it is based on OpenFOAMs *macro expansion*
- Both mechanism are done only once during calculation
  - at slightly different times

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Macro expansion in OpenFOAM

Macro expansion usually starts with $

Simple macro expansion

- in the simplest case the $ is followed by a `name`
- the value of the `name` is copied over

```
a inlet;
b $a;      // also "inlet"
```

- this is only done when the dictionary is read
  - if it is rewritten there will be no `$a` but `inlet`

"Copy and modify"

- There is a mechanism for dictionaries to "inherit" from others

In functions in controlDict

- Name of a dictionary between $ and ; pulls in the whole dictionary
- Subsequent entries "overwrite" the original values

```
TValues {
   type swakExpression;
   valueType internalField;
   expression "T";
   verbose true;
   accumulations (
     min
     weightedAverage
     max
   );
}
kineticEnergy {
  $TValues;
  expression "rho*U&U";
}
```

Advanced dictionary access

The mechanisms so far only work on the same dictionary level

Accessing dictionary values and traversal

- . and : after the $ change the level that is accessed

```
a 10;
dict {
    b $..a;   // go up one level
    subdict {
        c $...a;   // two levels
        d $:a;     // top level
    }
}
e $dict.b;         // access sub-dictionary
```

Including other files

- #include pulls in the content of another files
  - only as text. Parsing happens as a whole
- This allows "reuse" of dictionary content across files

bcValues

All boundary conditions in one place

```
TWall 300;
UWall (0 0 0);
```

T

Using in one field file

```
#include "bcValues";
internalField uniform $TWall;
boundaryField {
    wall {
        type fixedValue;
        value $internalField;
    }
}
```

It all happens only once

- Macro expansion is more flexible than C++ macro-expansion
    - Knows about the structure of the file
- Still it is not a link
    - The value at the first evaluation "sticks"
        - If the original changes the "expanded" value stays the same
        - Sometimes it would be nice to have it change as well
        - But usually the current behavior is better (think `$internalFields` in boundary conditions)

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. **Before the evaluation**
   - OpenFOAM macro expansion
   - **swak macro expansion**
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Does swak need its own expansion?

- No
  - But it is nice to have
- `swak4Foam` macro expansion happens after OpenFOAM macro expansion
  - But before the first evaluation happens
    - While the string is read
  - Done for all strings that hold expressions
    - `variables` etc
- It is also triggered by presence of $ in the expression strings
  - Done until there is no more $ in the string
- The resulting string is stored in memory
  - It is the one seen during error messages

Simple values

- This is done for variables that are on the same level as the expression string
- The sub-string `$name` is replaced by the value of the variable `name`
    - The name may only consist of letters, digits and _
        - May not start with a digit
        - The first non-matching character is terminating the name

Getting the density

```
kineaticEnergy {
    rho 1.245;
    expression "$rho*U&U";
}
```

More complicated values

- if things like dictionary traversal are needed the macro has to be written like this: `$[macro]`
  - The regular macro expansion `$macro` is done
  - The result is placed into the expression string

Getting the density from the top

```
rho 1.245;
kineticEnergy {
    expression "$[:rho]*U&U";
}
```

Casting special values

- Sometimes the expanded value is not a valid `swak4Foam`-expression
  - In such cases it has to be "cast" to the desired type
    - This includes `vector` (`(1 2 3)` to `vector(1,2,3)`) and `tensor`
  - Syntax is similar to C-casting: `$[(type)macro]`
    - Get `$macro` and interpret is as `type`
    - Most common types are implemented
    - Complete list of types can be got by setting `type` to `banana`

Getting the dimensioned density

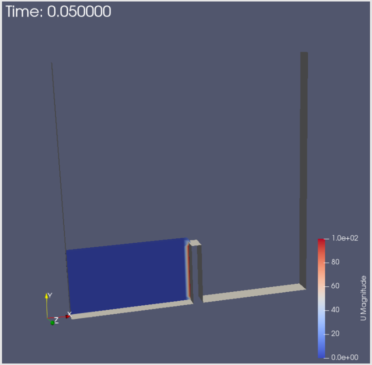
```
rho rho [1 -3 0 0 0 0 0] 1.245;
kineticEnergy {
    expression "$[(dimensioedScalar):rho]*U&U";
}
```

Example: non-breaking dam

Description

- The geometry from the `damBreak`-tutorial is reused
- Modifications:
  - One "basin" is filled with water
- No water column
  - Walls of the obstacle in the middle are "conveyors"
    - Against the gravity direction
    - Only component "parallel" to gravity
  - Conveyors switch off after 60% of the simulation time

Geometry



Getting gravity direction

We want to reuse the gravity that is already there

constant/g

```
dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.81 0);
```

0/U

- Including into sub-dictionaries doesn't "pollute" the dictionary as much
  - In this case we also avoid a "clash" of the dimensions

```
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);

g {
    #include "$FOAM_CASE/constant/g"
}

control {
        #include "$FOAM_CASE/system/controlDict";
}
```

The boundary condition

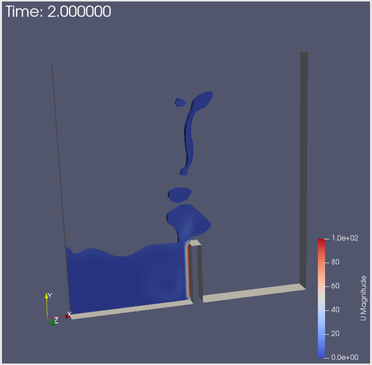
- Gets the down-direction by normalizing the gravity vector
- removes the component perpendicular to the wall
- Switches off after 60% of the run-time

0/U

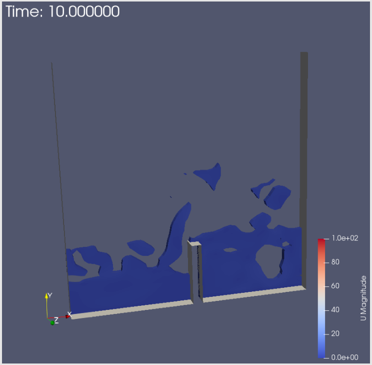
```
lowerWall
{
    type groovyBC;
    value $internalField;
    valueExpression "-doIt*100*down*(alphaW>0.1␣?␣1␣:␣0)";
    variables (
        "g=$[(vector):g.value];"
        "down=g/mag(g);"
        "normalPart=normal()␣&␣down;"
        "down=down-normal()*normalPart;"
        "end=$[:control.endTime];"
        "doIt=time()<end*$endRatio␣?␣1␣:␣0;"
    );
    aliases {
        alphaW alpha.water;
    }
    endRatio 0.6;
}
```

Result: non-breaking

First splash



Aftermath



Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Special needs

- swak4Foam has a lot of functions built in
  - common mathematical functions like sin, cos to . . . . besselJ0, erf
  - information about the discretization like pos(), vol()
  - to more esotheric like distToPatch
- But sometimes more special stuff is needed like
  - Information about the mesh-quality
  - Reaction rates
  - Calculated properties of the turbulence/thermophysics
  - Number of cells to the outlet
- Adding all these to the parser
  - would pollute the namespace
  - bloat the libraries
  - make it hard to maintain the parsers

Function plugins

- The solution are *function plugins*
  - libraries that can be loaded at run-time
  - add special functions to special lookup tables
    - these functions can be used in the parsers like built-in functions
- a number of function-plugins come with `swak4Foam`
  - a incomplete list will follow
- additional function plugins can be written yourself
  - use the existing ones as examples
  - the most complicated part is declaring the parameters and the return value
    - and "registering" the functions
  - if a environment variable `SWAK_USER_PLUGINS` is specified then these will be compiled by the regular `Allwmake` of `swak4Foam`
    - Content of the variable would be the locations of the library sources separated by `;`
- by convention the names of the libraries
  - start with `libswak`
  - end with `FunctionPlugin.so`
  - the part between that is the *name* of the function plugin
    - the name of `libswakMeshQualityFunctionPlugin.so` is `MeshQuality`

How to use

There are two ways to use function plugins

controlDict

This introduces them for the whole project

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

funkySetFields

Sometimes a function is only needed for post-processing

- funkySetFields has an option to load function plugins
  - parameter is a comma-separated list of the plugin names

```
> funkySetFields -functionPlugins MeshQuality,MeshWave -time 0 -create -field ortho -<brk>
      <cont>expression "faceAverage(mqFaceNonOrtho())"
```

Listing available functions

- if function plugins are load `swak4Foam` gives a complete list in the beginning
  - unfortunately this currently is all the documentation there is

on the output

```
"Loaded␣plugin␣functions␣for␣'FieldValueExpressionDriver':"
  cellColouring:
    "volScalarField␣cellColouring()"
  floodFillFromCells:
    "volScalarField␣floodFillFromCells(internalField/volLogicalField␣blockedCells)"
  floodFillFromFaces:
    "volScalarField␣floodFillFromFaces(internalField/surfaceLogicalField␣blockedFaces)"
  meshLayersFromCells:
    "volScalarField␣meshLayersFromCells(internalField/volLogicalField␣blockedCells)"
  meshLayersFromFaces:
    "volScalarField␣meshLayersFromFaces(internalField/surfaceLogicalField␣blockedFaces)"
  meshLayersFromPatch:
    "volScalarField␣meshLayersFromPatch(primitive/word␣patchName)"
  mqCellAspectRatio:
    "volScalarField␣mqCellAspectRatio()"
  mqCellFaceNr:
    "volScalarField␣mqCellFaceNr()"
  mqCellShape:
    "volScalarField␣mqCellShape()"
  mqFaceNonOrtho:
    "surfaceScalarField␣mqFaceNonOrtho()"
  mqFaceSkewness:
    "surfaceScalarField␣mqFaceSkewness()"
```

Explanation of the function descriptions

- Each parser has a separate list
    - One line `Loading plugin functions for`
- Description of a function consists of two lines
    1. the name of the function
    2. calling convention
        1. type of the return value
        2. name of the function (again)
        3. List of the parameters inside ()
- Description of a parameter consists of
    1. type of sub-parser used for this parameter
    2. a /
    3. expected type of the value returned by the sub-parser
    4. a descriptive name for the parameter

Sub-parser

- the sub-parser can be either
  - a `swak4Foam` parser
    - this is a full parser (uses all parameters and can have sub-parsers)
  - `primitive` indicates that a simple (OpenFOAM) value is read
    - like `scalar`, `word`, `string`
    - OpenFOAM is used to parse it
- the sub-parser is used until an unmatched , or) is found
- if the sub-parser fails then it is a problem of the sub-parser
  - but the problem escalates to the "parent" parser
  - error messages are printed as a "stack"
    - the sub-parsers at the bottom

Stacked error messages

- floodFillFromCells expects a logical expression
  - result is the *source region*
  - cells that can be reached from the *source region* are marked
  - purpose: mark unconnected regions of the mesh
- if the parser doesn't end with a logical expression then it fails

Sub-parser fails

```
> funkySetFields -functionPlugins MeshWave -time 0 -create -field fromTop -expression "<brk>
      <cont>floodFillFromCells(pos().y<0)"
... runs OK
> funkySetFields -functionPlugins MeshWave -time 0 -create -field fromTop -expression "<brk>
      <cont>floodFillFromCells(pos().y)"
....
--> FOAM FATAL ERROR:
 Parser Error for driver FieldValueExpressionDriver at "1.8" :"syntax error, unexpected ')<brk>
       <cont>'"
"pos().y)"
        ^
--------|

Context of the error:


- Driver constructed from scratch
  Evaluating expression "floodFillFromCells(pos().y)"
  Plugin Function "floodFillFromCells" Substring "pos().y)"
- Driver constructed from scratch
  Evaluating expression "pos().y)"
```

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Discretization

- when using `div()` and `grad()` in the regular parser the settings from `systen/fvSchemes` are used
  - only explicit (`fvc`) implementations are used
- the `FvcSchemes` and `FacSchemes` plugins give direct access to the discretization schemes
  - bypasses `fvSchemes`
    - specification string is a parameter (needs `""` around it)
  - names are slightly different from the regular name
    - expected parameter value type is part of the name (`Scalar`, `Vector`, . . .)

Difference of schemes

It is quite instructive to see the difference between discretizations

```
> funkySetFields -functionPlugins FvcSchemes -time 100 -create -field diffUpwind -<brk>
     <cont>expression "fvcConvectionDivScalar(phi,fraction,\"Gauss upwind\")-<brk>
     <cont>fvcConvectionDivScalar(phi,fraction,\"Gauss linear\")"
```

Mesh Wave

- Functions that are based on the `MeshWave` algorithm in OpenFOAM
  - traverses the mesh and sets values based on this
  - works in parallel
- Function-"classes" are
  - `floodFill` for finding connected mesh regions
  - `meshLayersFrom` for finding the "discretization distance= from certain features
    - applications like "treating the 2 cell layers from the oulet differently"
  - `cellColouring` "colors" the cells so that no two neighbouring cells have the same colour
    - with a minimum number of colours
    - application: showing the mesh structure in ParaView
- Plugin name is `MeshWave`

Accessing physical models

- Plugins to access physical sub-models like
  - Transport properties
    - viscosity etc
  - Thermophysical models
  - Radiation
    - Absorpiton coeffs etc
  - Chemistry
    - Reaction rates etc
  - Turbulence
- Does so by looking for the model in the `objectRegistry`
  - Calling the appropriate methods
  - If called from `funkySetFields` they try to load this model
    - May not work everywhere
- These plugins give access to information that is always there
  - But OpenFOAM doesn't give voluntary access to it
- Plugin names are
  - `ThermoTurb`
  - `TransportTurb`
  - `RadiationModel`
  - `ChemistryModel`

Mesh Quality

- This plugin gives access to information that `checkMesh` only reports summarized
  - Orthogonality
  - Skewness
  - Aspect ratio
  - Cell shapes
- Uses the "original" functions
  - Some quantities are not easily post-processed because they are face-based
    - for instance the orthogonality
    - ParaView can't handle that
- Plugin name is `MeshQuality`

Local calculations

- This library does local calculations over the faces of cell
  - Stores the results per cell
- Originally introduced to make visualizations of `MeshQuality`-results possible
  - "orthogonality of the cell is the maximum orthogonality of its faces"
- Implemented calculations are
  - minimum
  - maximum
  - average
- Plugin name is `LocalCalculations`

Velocity and mesh movements

There are plugins concerned with movement in different forms

Velocity functions on the velocity field

- Courant numbers
- the stream-function

MRF the *moving reference frame* model

- make velocities absolute and relative

DynamicMesh properties of the mesh movement

- mesh Courant number
- mesh velocity and flow

Getting discrete to continuous

Plugins that "project" discrete structures to continuous fields

LagrangianCloudSources influences of a cloud on the continous phase

- mass and volume fraction of the cloud
- source terms for equations (momentum, mass, energy, ...)

SurfacesAndSets sampled surfaces and sets in the continuous phase

- "does the set/surface touch this cell"
- "how big is the relative area of the surface in this cell"
- "how big is the distance of the set/surface to this cell"
- ...

Spatially shifting values

- The plugin ShiftField allows shifting the value of fields
  - Application "Give me the temperature 2 meters from this place"
- Different variations
  - How the shift vector is calculated (constant or calculated)
  - How areas "ouside" should be treated (default)
- This relies on the mesh interpolation of the underlying OpenFOAM-fork
  - Quality of results differ

Which quantile is the current cell in

- The plugin is named `Quantile`
    - helps finding things like "the hottest 10% of the geometry"
- Calculates the distribution function of the field
- Then reports for each cell how many percent of the volume is smaller than this
    - Also allows comparing to a different distribution
        - Example: comparing the temperature of two different phases

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Zones and sets

- OpenFOAM has two kinds of "subsets" for meshes
  - Zones
    - can not change
    - mutual exclusive
    - loaded automatically when mesh is loaded
  - Sets
    - can change their content
    - a cell can belongto more than one set
    - loaded when needed
- These exist for
  - cells
  - faces
  - points
- `swak4Foam` has one parser for all of them
  - called the *Subset parser*
  - the drivers are different
  - drivers exist for
    - cell zones and sets
    - face zones and sets
    - not for points (never needed it)

Restrictions

- The *subset parser* has no secondary data structure
  - what would that be? "the faces of a cell set"?
  - OpenFOAM has no support for it
    - So it would mean a complete reimplementation
- One parser for multiple drivers means that there are undefined functions
  - For instance: `vol()` is not defined for `faceZone`
    - If you call it the expression will fail
  - If such an inappropriate function is called the driver fails

Interpolation for faces

- `faceSet` and `faceZone` do their calculations on the faces
- Hardly any values in OpenFOAM are defined on the faces
    - Most notable exception: the flux `phi`
- So hardly anything of interest could be calculated there
- `swak4Foam` can interpolate cell values to the faces
    - But it doesn't so automatically
        - "Principle of least surprise"
    - Has to be switched on by the `autoInterpolate` option
        - Otherwise it fails (because the field can not be found)
    - Still issue a warning every time it interpolates
        - Can be switched off by `warnAutoInterpolate`

Orientation of faces

- when using things like `phi` on `faceSet` or `faceZone` it is not sure that correct results are calculated
  - because some faces might be oriented differently and then the sign of `phi` is "wrong" there
- for this exists the variable `flip()`
  - 1 for "correctly" oriented faces
    - correctly" is a question of definition
  - -1 for others
- for `faceZone` the value of `flip()` is "defined" and set by the OpenFOAM-utilities
- for `faceSet` the default is 1
  - unless an appropriately named `cellSet` is found
    - for a `faceSet` named `foo` the name would be `fooSlaveCells`
  - then `flip()` is calculated in such a way that `flip()*face()` points away from these cells

Statically creating sets and zones

- the library `libswakTopoSources.so` adds new topological sources
  - Can be used everywhere these are used to add entities based on expressions
    - If the logical expression evaluates to `true` then the cell/face/point is part of the set/zone
    - Special case `face`: if the expression is defined on the cells then the boundary between `true` and `face` is used

system/topoSetDict in other/topoSetDam

```
actions (
    {
        type faceSet;
        name middleFaces;
        action new;
        source expressionToFace;
        sourceInfo {
            expression "pos().x>0.291";
        }
    }
    {
        type cellSet;
        name centerCells;
        action new;
        source expressionToCell;
        sourceInfo {
            expression "mag(pos()-vector(0.291,0.291,0.007))␣<␣0.1";
        }
    }
);
```

Loading sets

- cellSet, faceSet, pointSet are only loaded by code that needs them
  - Then they are also registered with the objectRegistry
- If we want to use them in a swak-expression we got to load them
  - There is a function object for that
    - Loads sets and registers them at the objectRegistry

system/controlDict in other/topSetDam

```
getTheSets {
    type loadTopoSets;
    forceLoading true;
    loadCellSets true;
    loadFaceSets true;
    loadPointSets false;
    writeSets true;
}
```

Calculating on the sets

- Once loaded the sets can be used for calculation

system/controlDict in other/topSetDam

```
middleLiquid {
    type swakExpression;
    valueType faceSet;
    setName middleFaces;
    aliases {
        aWater alpha.water;
    }
    verbose true;
    expression "aWater";
    accumulations (
        min
        weightedAverage
        max
    );
    autoInterpolate true;
    warnAutoInterpolate false;
}
```

Dynamically creating and manipulating sets

- There function objects for that

system/controlDict in other/topSetDam

This collects all "non-pure" cells and calculates their velocity

```
undecidedCells {
    type manipulateCellSet;
    cellSetName undecided;
    aliases {
        aWater alpha.water;
    }
    mask "0.1<aWater␣&&␣aWater<0.9";
    createMissing true;  // create set if it is not already there
    outputControl timeStep;
    outputInterval 1;
}
undecidedVelocity {
    $middleLiquid;
    valueType cellSet;
    setName undecided;
    expression "mag(U)";
    accumulations (
        size
        weightedAverage
        max
    );
}
```

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- **Sets and surfaces**
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Sampled sets and surfaces

- *Sampled sets* are collections of points on which values can be collected during time
    - They used to be called `probes`
- *Samples surfaces* are surfaces on which values can be collected
    - Can be defined in various ways
        - Pure geometric specification
        - in relation to patches
        - as iso-surfaces of a value
    - Advantage compared to `faceZone` and `faceSet`: doesn't have to be aligned to the mesh
        - Disadvantage: computationally "expensive"

A repository of their own

- Sampled `set` s and `surface` s are mostly used in function objects of the same name
  - That is why there are not registered in the `objectRegistry`
    - Which makes it hard for `swak4Foam` to access them
- `swak4Foam` introduces their own registry for it
  - sets and surfaces from the "regular" function objects are unfortunately not registered there
  - but sets and surfaces registered there can be reused
    - just specify `type swakRegistryProxy;`
    - needs `setName` for sets
    - `surfaceName` for surfaces
- Information about these repositories is automatically written at write time
  - Repository handles writing of these sets and surfaces as well
    - if `autoWriteSurface` (or `Set`) is specified
    - needs `surfaceFormat` (or `set`) to be specified
    - writing at creation can be forced with `writeSurfaceOnCreation`

Creating them with a function object

controlDict of FromPresentations/OSCFD_cleaningT

Create (and update) a surface that is at the water/air interface

```
createInterface
{
    type createSampledSurface;
    outputControl timeStep;
    outputInterval 1;
    surfaceName interface;
    surface {
        type isoSurfaceCell;
        isoField fraction;
        isoValue 0.1;
        interpolate true;
    }
    writeSurfaceOnConstruction true;
    autoWriteSurface true;
    surfaceFormat vtk;
}
```

later in the same file

Sample at a single point (the sensor location)

```
createMeasurment
{
    type createSampledSet;
    outputControl timeStep;
    outputInterval 1;
    setName sensor;
    set {
        type cloud;
        axis x;
        points (
            (0.45 0.1 0.025)
        );
    }
    writeSetOnConstruction true;
    autoWriteSet true;
    setFormat vtk;
}
```

Values on surfaces

Interpolation

- volume fields can be used as usual
    - although not really "defined" on the surface
- Each function object has to specify with `interpolationType` how the values should be sampled

controlDict of other/capillaryRise

Report the velocity of a surface

```
velocity
{
    type swakExpression;
    valueType surface;
    surfaceName interface;
    verbose true;
    expression "mag(U)";
    accumulations (
        max
    );
    interpolationType cell;
}
```

Surface properties

Properties of the surface

- surfaces and sets have special functions to access the properties of their components
  - `pos()` for the positions
  - `area()` for the sizes of the triangles
  - `normal()` for the normal vector

`controlDict` of `FromPresentations/OSCFD_cleaningT`

Height (assuming y is "up") of the interface

```
height
{
    type swakExpression;
    valueType surface;
    surfaceName interface;
    verbose true;
    expression "pos().y";
    accumulations (
        min
        max
        size
    );
    interpolationType cellPoint;
}
```

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Lagrangian particles

- Lagrangian parsers are organized in a separate library
  - Their implementation is a bit special because many interesting properties are only accessible through C++-code
    - These calls differ between particle classes
    - And between OpenFOAM-version
  - For every known particle class `swak4Foam` implements a `CloudProxy` that handles these calls
    - `particleCloud`
    - `kinematicCloud`
    - `thermalCloud`
    - `reactingCloud`
  - `swak4Foam` automatically selects the appropriate proxy for a cloud
  - for other clouds the user would have to write an *adaptor class*
- Because nobody wants to look at the source code a list of available functions is output when a cloud-parser is created
  - Includes short descriptions
    - Function names are usually the ones from the original C++-API
    - Beware: some are defined for `particles` some for `parcels`. Like the C++-API

List of properties

Output when a cloud parser is constructed

```
Driver for cloud dirt of type Cloud<basicKinematicParcel> (Proxy type: CloudProxy)
     List of functions:
                Name |            Type | Description
--------------------------------------------------------------
                   U |          vector | Velocity
               UTurb |          vector | Turbulent velocity fluctuations
              active |            bool | Is this parcel active?
                 age |          scalar | Age of the prticle
               areaP |          scalar | Particle projected area
               areaS |          scalar | Particle surface area
                cell |          scalar | number of the cell
 currentTimeFraction |          scalar | Current fraction within the time-step
                   d |          scalar | Diameter
             dTarget |          scalar | Target diameter
                face |          scalar | number of the face
                mass |          scalar | Particle mass
         minParcelMass |          scalar | Minimum parcel mass (constant)
           nParticle |          scalar | Number of particles
          onBoundary |            bool | is this currently on the boundary
      onBoundaryFace |            bool | is this currently on the boundary
      onInternalFace |            bool | is this currently on the internal
              origId |          scalar | Original id
            origProc |          scalar | Originating processor
                 rho |          scalar | Density
                rho0 |          scalar | Particle density (constant)
              rhoMin |          scalar | Minimum density (constant)
        stepFraction |          scalar | fraction of the time-step completed
               tTurb |          scalar | Time in turbulent eddy
              typeId |          scalar | Type ID
              volume |          scalar | Particle volume
```

Interpolating from the continuous phase

- There is a function `fluidPhase` that gets the value of a fluid phase field at location of the particle
  - For instance `T-fluidPhase(T)` gives the difference of the temperature of a *thermal parcel* to the surrounding temperature
- an optional dictionary `interpolationSchemes` specifies which interpolation is to be used for the field `T`
  - otherwise the corresponding dictionary from the cloud specifiation file in `constant` is used

Other lagrangian stuff

There are two libraries with cloud function objects

swakCloudFunctionObjects Currently only has `eliminateBySwakExpression`

- eliminates parcels if an expression evaluates to `true`

simpleCloudFunctionObjects Gunction objects that are mainly for diagnosing/fixing problems in the tracking algorithm

- statistics of the number of faces particles crossed/collided with etc
- eliminate parcels that were caught in infinitely little rebounds
- tracing the paths (with all properties) of selected parcels

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Function1 / DataEntry with swak4Foam expressions

- Most OpenFOAM-Versions have a data structure called `Function1`
  - Used to be called `DataEntry`
- Represents a single value that depends on another variable
- Frequently used in boundary conditions
  - Like `flowRateInletVelocity`
    - Value is the flow rate
    - Variable is the time
- Run-time selectable
  - `constant`
  - `table`
  - . . .
- `swak4Foam` adds an implementation that allows using an expression for this
  - Configured by a dictionary

    expression the actual expression

    independentVariableName the name of the variable in the expression

    valueType where the expression is evaluated. All other parameters (`patchName`, `variables` . . .) depend on that

Adjustable mass-flow

- In this example the volume flow is ramped up to the proper value
  - This sometimes avoids instabilities at startup

system/controlDict

- If no other swak4Foam things (function objects, boundary conditions ..) are used at run-time then a special function object has to be added
  - this has technical reasons

```
functions {
    initSwak {
        type  initSwakFunctionObject;
        region  region0;
    }
}
```

boundary in 0/U

```
inlet
{
    type              flowRateInletVelocity;
    volumetricFlowRate         swak {
        expression "t<1␣?␣0.1*t␣:␣0.1";
        valueType patch;
        patchName inlet;
        independentVariableName t;
    };
    value             uniform (0 0 0);
}
```

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Variables

- `variables` are one basic tool of `swak4Foam`
  - allow splitting calculations into smaller parts
- the format is a list of strings
  - parts of a string are
    1. variable name
    2. = (assignment operator)
    3. expression
    4. ; (termination)
- behavior of the variables can be adapted by
  - changing the variable to an external
  - listing it as special

External expressions

- Regular variable assignment

```
varName=expression;
```

- External expressions are triggered by {}

```
varName{parserType'name/regionName}=expression;
```

parserType parser type (patch, internalField etc)

name specification for the parser (for instance the *patch name* to calculate on for patch)

regionName the mesh region to use for multi-region cases

- The value is calculated *remotely* but used *locally*
  - Restriction: because there is no general way to interpolate expression must yield a uniform value (min, max, average, sum)
- Simplifications:
  - When calculating on the same mesh regionName can get lost

```
varName{parserType'name}=expression;
```

- If no ' is found it is assumed that parserType is patch

```
varName{patchName}=expression;
```

The classic: pressure drop

This is the most-used external expression

functions in controlDict

Calculating the pressure drop

```
pressureDrop
{
    type swakExpression;
    valueType patch;
    patchName inlet;
    verbose true;

    variables (
         "pOut{patch'outlet}=sum(p*area())/sum(area());"
    );
    accumulations (
        weightedAverage
    );

    expression "p-pOut";
}
```

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Why globals?

- Variables are local to each entity
  - boundary conditions
  - function objects
- Sometimes there is a need to make data available to other entities
  - There are function objects that rely in this mechanism for making their results available
    - Especially the scripting languages support

Global variables

- To move data from one function object to another swak4Foam has something called *Global variables*
- To have some kind of separation they are organized in namespaces
  - Organize the variables into namespaces by "topic"
    - In our case solver for solver data
- Function objects that can write global variables have an entry toGlobalNamespace
- Everywhere where you can specify variables you can add an optional globalScopes
  - This is a list with names of global namespaces
  - All the variables in these namespaces are "injected" before the regular variables
  - Attention: the size of the global variables must match the size of the entity (for instance: number of faces)
    - If the variable is "uniform" it matches anywhere

Creating global variables

controlDict of FromPresentations/OSCFD_cleaningTank2D

```
defineState {
    type addGlobalVariable;
    outputControl timeStep;
    outputInterval 1;

    globalScope outletState;
    globalVariables {
        closed {
            valueType scalar;
            value 0;
            isSingleValue yes;
        }
        airReachedOutletTime {
            valueType scalar;
            value -1;
            isSingleValue yes;
        }
        shutdownTime {
            valueType scalar;
            value 1;
            isSingleValue yes;
        }
    }
}
```

Using global variables

controlDict of FromPresentations/OSCFD_cleaningTank2D

```
openIfSensorReached {
    type calculateGlobalVariables;
    valueType set;
    setName sensor;
    toGlobalNamespace outletState;
    globalScopes (
        outletState
    );
    set {
        type swakRegistryProxy;
        axis y;
        setName sensor;
    }
    toGlobalVariables (
        closed
        airReachedOutletTime
    );
    variables (
        "state=average(alpha1);"
        "thresA=0.9;"
        "opening=(closed>0.5␣&&␣state>thresA)␣?␣1␣:␣0;"
        "closed=(opening>0.5)␣?␣0␣:␣closed;"
        "airReachedOutletTime=(opening>0.5)␣?␣-1␣:␣average(airReachedOutletTime);"
    );
    aliases {
        alpha1 alpha.water;
    }
}
```

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Stored variables

- Regular entries in `variables` forget their values between time-steps
- When we specify them in the `storedVariables`-list they don't
  - They are even saved and read on restart
- Specification of a stored variable needs two things

name	
intialValue	the value that should be used when the variable has never been set before

- When the variable is on the right of a = the stored value is used
- The last value the variable is set to is stored for the next time-step
- `storedVariables` are aware that there can be multiple iterations per time-step
  - old values are from the last time. Not the last iteration

Remembering the biggest value

system/controlDict in groovyBC/wobbler

```
biggestDFreeMem
{
    type patchExpression;
    patches (
        free
        forced
    );
    storedVariables (
        {
            name maxD;
            initialValue "0";
        }
    );
    variables ( "maxD=(␣mag(D)␣>␣maxD)␣?␣mag(D)␣:␣maxD;");
    accumulations (
        max
    );
    expression "maxD";
    verbose true;
}
```

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - **Delayed expressions**
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Delayed variables

- Delayed variables are special variables with a schizophrenic behaviour
  - When written to they behave like regular variables
  - When read they don't use the current value but the value set some time ago (the *delay*)
- They are declared in a list `delayedVariables` of dictionaries

  name the name under which the variable is known

  delay how far back in time it should go

  startupValue during the first `delay` seconds there is nothing to remember. This value is used instead

  storeInterval this is the interval at which values should be remembered. When remembering values between that are interpolated
    - set it too high: you might run out of memory
    - set it too low: it might be inaccurate
    - in a steady simulation 1 means: we remember everything
- Values longer ago than `delay` are forgotten

What to use them for

- Expression where an input triggers a delayed reaction
    - and we don't want to (or can't) model the whole system that causes the delay
- In the example below it is a pump that recycles our liquid
- But it can be a sensor that has a switching time

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

Mapping in OpenFOAM

- OpenFOAM offers a mechanism called "mapped" patches
  - Usually used for multi-region cases
- Values from one patch is mapped to the other
- *mapped patches* have to be declared in `constant/polyMesh/boundary`
  - `blockMesh` knows how to do that

blockMeshDict

```
inlet1
{
    type            mappedPatch;
    offset          ( 0 0.25 0 );
    sampleRegion    region0;
    sampleMode      nearestFace;
    samplePatch     none;
    faces (
        (1 5 4 0)
    );
}
```

polyMesh/boundary

```
inlet1
{
    type            mappedPatch;
    inGroups        1(mappedPatch);
    nFaces          20;
    startFace       11640;
    sampleMode      nearestFace;
    sampleRegion    region0;
    samplePatch     none;
    offsetMode      uniform;
    offset          (0 0.25 0);
}
```

Swak-functions for mapping

swak4Foam supports mapped patches in 2 ways

1. in the patch parser

   mapped(fieldName) gets the value of the field from the mapped partner patch

   mappedInternal(fieldName) gets the internal field

2. in external expressions
   - in var{patchName}=expression; the expression doesn't have to be uniform if patchName is the mapped partner patch of the current patch

Example: mapping channel

- inlet2 is outlet1 minus 1
- inlet3 is supposed to be outlet2

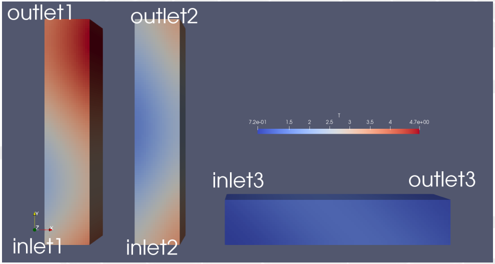


Figure: Transport in channels with uniform mapping. Case: `tests/mappingChannels`

Non-uniform distances

- Previous picture illustrates a problem
  - Out of the box OpenFOAM only allows uniform offset between patches
- swak4Foam has a utility calcNonUniformOffsets that calculates rotated/scaled offsets
  - writes them into the boundary file

calcNonUniformOffsetsDict that drives the utility

Translation, rotation and scaling are allowed

```
offsetSpecifications {
                     inlet3 {
                         mode specifyAll;
                         transposeFirst  ( -0.4 -0.05 0 );
                         scaleBeforeRotation (1 1 1);
                         rotationFrom (0 1 0);
                         rotationTo (-1 0 0);
                         scaleAfterRotation (1 1 1);
                         transposeAfter ( 0.25 0.5 0);
                     }
}
```

Example: non-uniform mapping channel

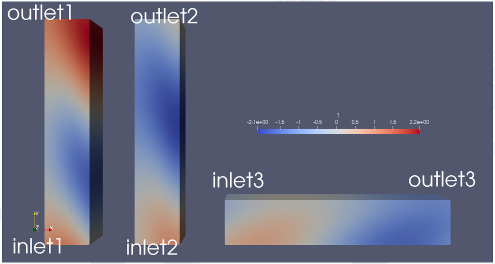


Figure: Transport in channels with one non-uniform mapping. Case: `tests/mappingChannelsNonUniform`

Outline

1 Introduction
- This presentation
- Who is this?
- swak4Foam

2 Parser explained
- General
- Native vs secondary
- Uniform
- My information is not there

3 Before the evaluation
- OpenFOAM macro expansion
- swak macro expansion

4 Function plugins
- Function plugins
- Implemented function plugins

5 Other parser
- Zones and sets
- Sets and surfaces
- Particles
- Other topics

6 Self-reference
- External expressions
- Global variables
- Stored variables
- Delayed expressions
- Mapped values
- Using it all: cleaning Tank

7 Conclusions

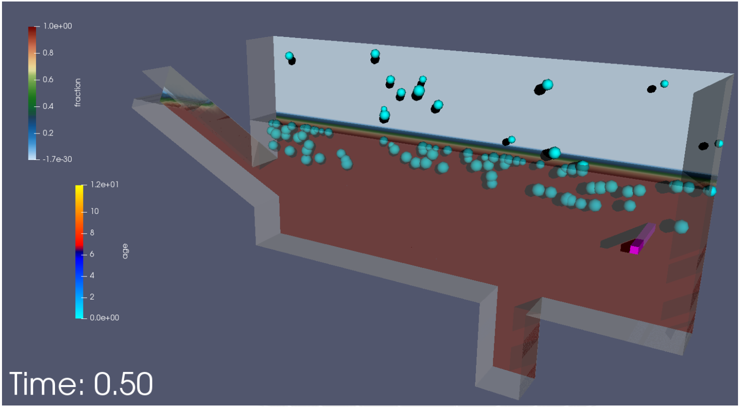
History of the case

- This case was first demonstrated at the OSCFD-conference 2012 in London
  - to demonstrate advanced capabilities of `swak4Foam`
- Now has been slightly modified
- Uses a lot of global variables to communicate states
  - Now that can be done simpler with *state machines*
    - See the presentation aboiut that from the Exceter Workshop 2017

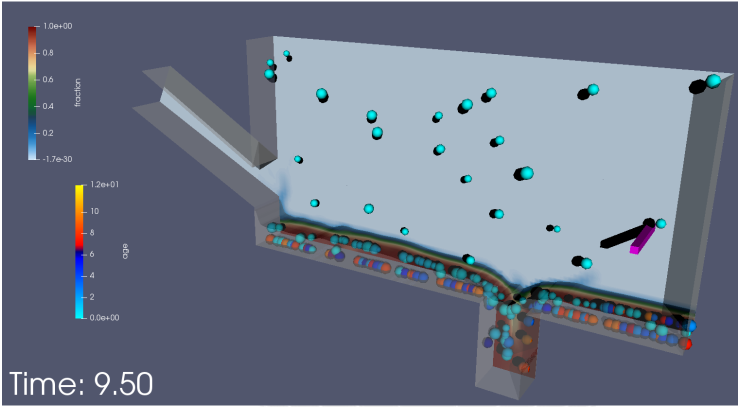
Description

- Dirt particles (lagrangian!) fall into a tank
    - Should be filtered out
- Water is let out of a tank
    - Until the water surface reaches the outlet (evaluations on a sampled iso-surface!)
    - Then the outlet closes
- The water from the outlet is pumped to an inlet
    - This needs 10 seconds (delayed variable!)
- A sensor is modeled by a sampled set
    - Once the water level reaches it the outlet re-opens
- Particles that reach the outlet are considered filtered
    - Cloud function object with expression
- This is repeated until all particles are gone
    - Dirt stops falling into the tank after 75 seconds

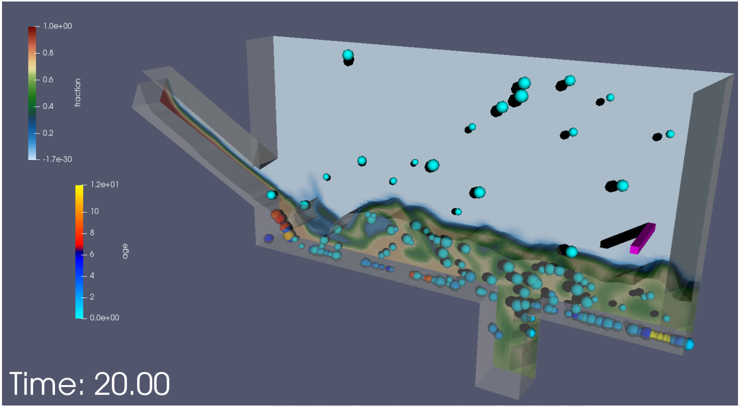
Initial conditions



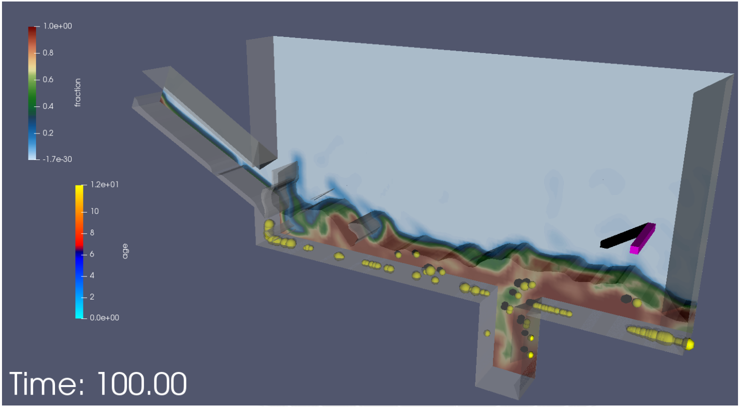
Emptying


1.0e+00
0.8
0.6
0.4
0.2
-1.7e-30
fraction
1.2e+01
10
8
6
4
2
0.0e+00
age
Time: 9.50

Refilling


1.0e+00
0.8
0.6
0.4
0.2
-1.7e-30
fraction
1.2e+01
10
8
6
4
2
0.0e+00
age
Time: 20.00

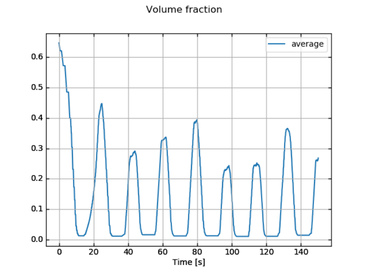
Almost cleaned



Where does the water go

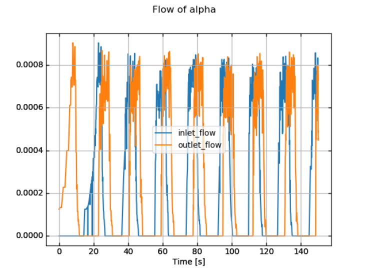
Water during the simulation

Water in the tank



Flows in and out

Notice how the inflow "follows" the outflow

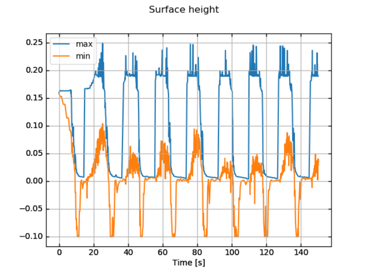


Heights - Fluid and Dirt

Height of water/air interface

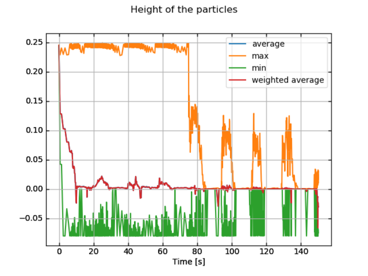
The y-component of the interface

- Droplets mess up the maximum

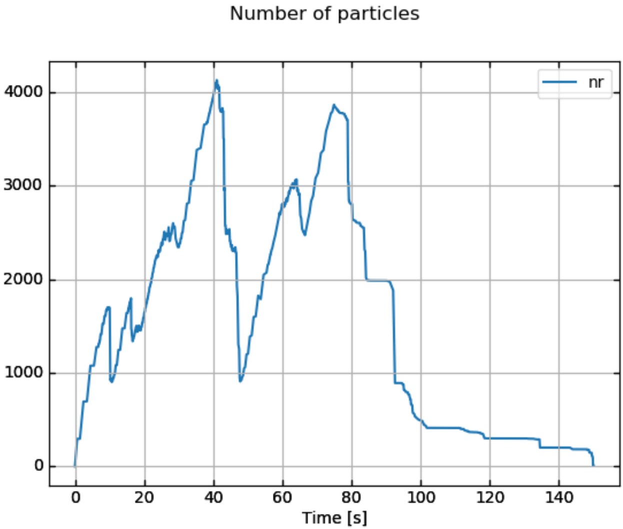


Where are the particles

The y-component of the particle locations



Result: Particles removed



Don't overdo it

- swak4Foam allows you to make interesting calculations
  - some of them take longer than the actual solution
- The ESI fork and the foam-extend-fork have facilities to calculate profiling info
  - Have to be activated
  - Write at each timestep to uniform/profilingInfo
- The utility pyFoamListProfilingInfo.py allows you to analyze that info
  - For "normal" computations the time spent in swak4Foam is less than 5%
  - In this thank-case it was more than 50%
- Use that info to decide which computations you actually need

controlDict

Adding profiling info in the ESI branch

```
profiling
{
    active      true;
    cpuInfo     true;
    memInfo     false;
    sysInfo     true;
}
```

Outline

1. Introduction
   - This presentation
   - Who is this?
   - swak4Foam
2. Parser explained
   - General
   - Native vs secondary
   - Uniform
   - My information is not there
3. Before the evaluation
   - OpenFOAM macro expansion
   - swak macro expansion
4. Function plugins
   - Function plugins
   - Implemented function plugins
5. Other parser
   - Zones and sets
   - Sets and surfaces
   - Particles
   - Other topics
6. Self-reference
   - External expressions
   - Global variables
   - Stored variables
   - Delayed expressions
   - Mapped values
   - Using it all: cleaning Tank
7. Conclusions

Goodbye to you

Thanks for listening

Questions?

License of this presentation

Authors of this document are:

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