swak4Foam

Gentle introduction and new developments

Bernhard F.W. Gschaider

Jeju, Korea
11. June 2013

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Outline

Outline

Aim of this presentation

- This presentation tries to give an overview of swak4Foam
- Shows how to enhance an existing case with it
- Discuss **some** of the most important concepts
- Introduce a **fraction** of the available functionality
- Tries not to bore you with canonical listings of functions
  - There is a documentation for that

Contents

- Basic usage of the most important features

  groovyBC flexible boundary conditions
  funkySetFields setting fields by expressions
  swakExpression function object with general expressions
- New features
  - Additional `searchableSurfaces` for `snappyHexMesh`
  - Calculating distributions

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

Previous presentations

- No C++, please. We're users! Held at the 6th Workshop in Penn State, 2011
  Similar in scope to this presentation. But with a different feature-set
- Automatic testing of solvers using PyFoam: Held at the Workshop in Darmstadt (2012)
  Despite the title also uses swak4Foam for evaluations
- swak4Foam - A toolbox for flexible case setup and evaluation, OSCFD 2012, London
  Describes several features of OpenFOAM (some not mentioned here) using one case

Sorry Trekies

- The two example cases in this presentation are inspired by the Star Wars movies
  - The "real" ones. Not the ones called Episode I-III
- It could have been about cases from the Star Trek-universe as well
  - But in Star Trek they run computer simulations on a regular basis and this works like this:
    - "Computer. If this class 42 star explodes how can the Enterprise ride the shockwave so that it will be catapulted past the Klingon homeworld on a course to Earth. Also I don't want my Earl Grey to be spilled and Ensign Crusher should be knocked unconscious" 3 seconds pause "Bling. Direct the enterprise to coordinates 13 666 4 switch off the Warp core and give me the keys to the Holodeck"
  - So obviously they don't have to bother with correct initial conditions, material parameters and boundary conditions and therefor have no need for swak4Foam

Outline

Command line examples

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

```
ls $HOME
```

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

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

Data files in PyFoamTraining_AdditionalFiles

- This is optional: adding some settings files
  - If you attended the PyFoam-training you've already got this
- A tar file with the required files in /home/openfoam/training_materials/Track3-2
  - unpack to home-directory

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

- Contents of the directory PyFoamTrainingMaterials

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

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

Adding stuff to the environment

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

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

- Install the files

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

Getting onto the same page

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

```
zsh
```

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

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

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

```
mkdir swak4FoamTraining; cd swak4FoamTraining
```

Outline

Why this section?

- I don't assume that all of you were present at the PyFoam-presentation before
- It is not necessary
  - We will "only" use it for running our cases

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 the two presentations mentioned above

Case setup

- Cloning an existing case

```
pyFoamCloneCase.py $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily test
```

- Decomposing the case

```
blockMesh -case test
pyFoamDecompose.py test 2
```

- Getting info about the case

```
pyFoamCaseReport.py test --short-bc --decomposition | rst2pdf >test.pdf
```

- Clearing non-essential data

```
pyFoamClearCase.py test --processors
```

- Pack the case into an archive (including the last time-step)

```
pyFoamPackCase.py test --last
```

- List all the OpenFOAM-cases in a directory (with additional information)

```
pyFoamListCases.py .
```

Running

- Straight running of a solver

```
pyFoamRunner.py interFoam
```

- Clear the case beforehand and only show the time

```
pyFoamRunner.py --clear --progress interFoam
```

- Show plots while simulating

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

- Change `controlDict` to write all time-steps (afterwards change it back)

```
pyFoamRunner.py --write-all interFoam
```

- Run a different OpenFOAM-Version than the default-one

```
pyFoamRunner.py --foam=1.9-beta interFoam
```

- Run the debug-version of the current version

```
pyFoamRunner.py --current --force-debug interFoam
```

Generated files

- Typically PyFoam generates several files during a run (the names of some of those depend on the case-name)

  case.foam Stub-file to open the case in ParaView

  PyFoamRunner.solver.logfile File with all the output that usually goes to the standard-output

  PyFoamRunner.solver..analyzed Directory with the results of the output analysis

  pickledPlots A special file that stores all the results of the analysis

  PyFoamHistory Log with all the PyFoam commands used on that case

Plotting

- Any logfile can be analyzed and plotted

```
pyFoamPlotWatcher.py --progress someOldLogfile
```

- A number of things can be plotted
  - Residuals
  - Continuity error
  - Courant number
  - Time-step
- User-defined plots can be specified
  - Specified in a file `customRegexp`
  - Data is analyzed using regular expressions
  - We will see examples for this later
- The option `--hardcopy` generates pictures of the plots

What else can PyFoam do for me?

- Write and read dictionaries from the command line
- Display the `blockMeshDict`
- Generate plots of Surfaces and sample lines
- Interact with paraView
- Control OpenFOAM-runs over the net

Outline

Outline

The pit of Carcoon

Figure : The infamous pit of Carcoon (source of picture: wikia.com. Fair use)

Feeding the Sarlacc

- As seen in The Return of the Jedi the Sarlacc living in the Great Pits of Carcoon (on the planet Tatooine) are a creature that Jaba-de-Hut feeds his enemies to
  - **This is wrong:** In reality the Sarlacc is a vegetarian creature that would never hurt a Jedi
    - The whole thing was a great misunderstanding
  - The main diet of the Sarlacc are oranges
    - These are placed on the surface of sand. The Sarlacc then sucks in the sand and digests the oranges
- The aim is to simulate this because Jaba-de-Hut wants to make sure that no oranges are wasted

Outline

Getting files for the basic case

- Now we get the files of the basic case
- First go to the right directory

```
cd $HOME/swak4FoamTraining
```

- Extract the case files

```
1 > tar xvzf /home/openfoam/training_materials/<brk>
    <cont>Track3-3/sandPit.baseCase.tgz
```

- Go to the case directory

```
cd sandPit.baseCase
```

Used solver

- For the following simulations we use `twoPhaseEulerFoam`
  - Heavy phase: sand
  - other phase: air
- Our case is based on the tutorial case `bed` with these modifications
  - the mesh is made wider (but the lower inlet keeps its width)
  - viscosity of the sand-phase is made larger to account for the specific properties of the Tatooine desert-sand

Running the case

This is a simple two-step:

- create the mesh

```
blockMesh
```

- use PyFoam to run the case

```
pyFoamPlotRunner.py --clear --progress twoPhaseEulerFoam
```

- then we look at the results
  - Sand settles from a low-density homogeneous distribution
  - Velocity fields are not very convincing when we're only interested in the movement of the sand

Velocity in the beginning

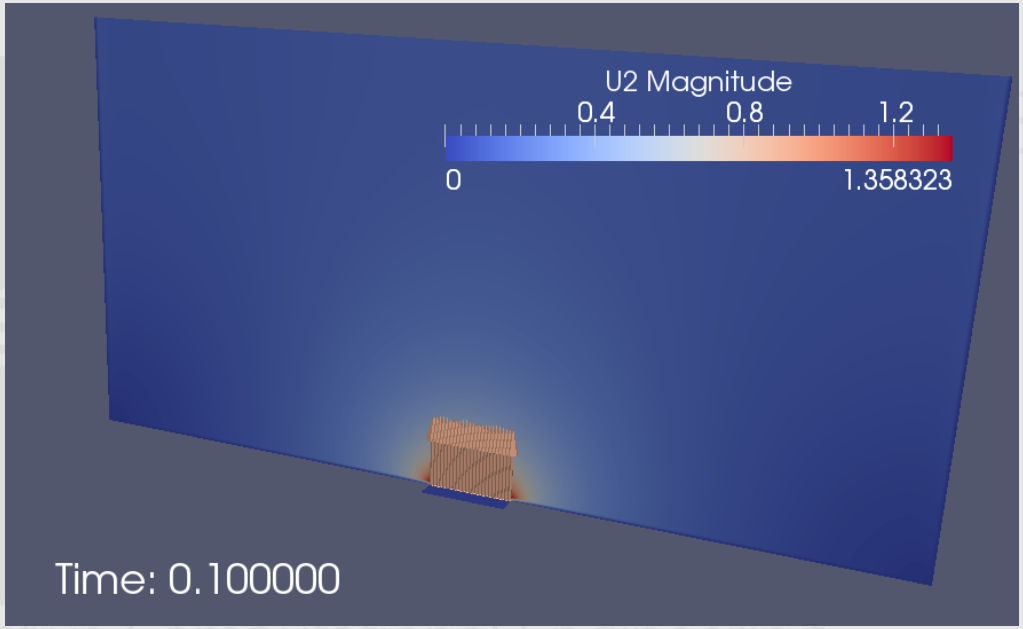
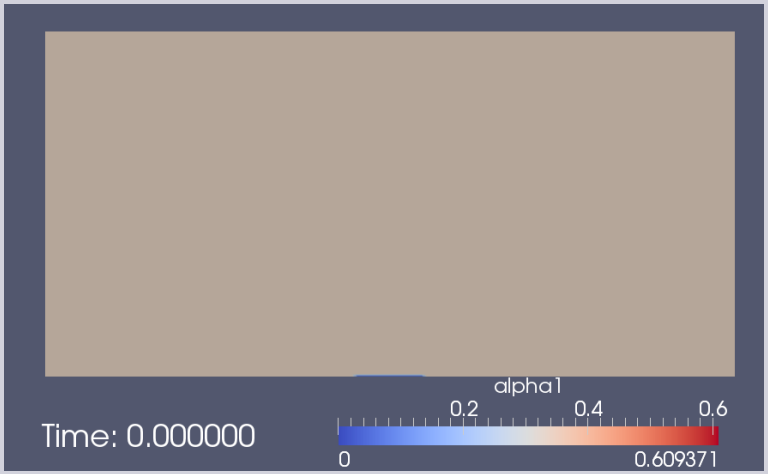


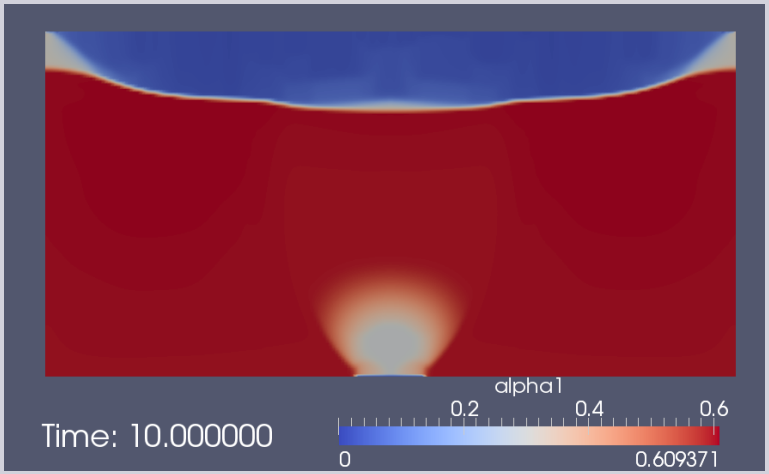
Figure : Velocity field and inlet velocity

Sand fraction

`alpha1` at the start



. . . and in the end



Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Sand velocity in the end

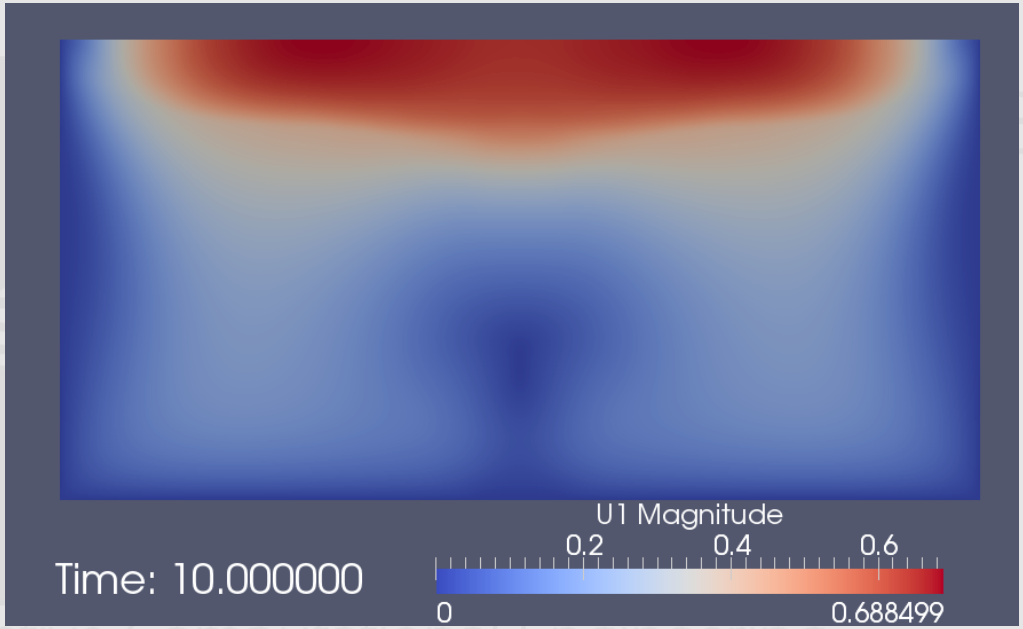


Figure : Sand velocity field in the whole domain

Weighted velocity in the end

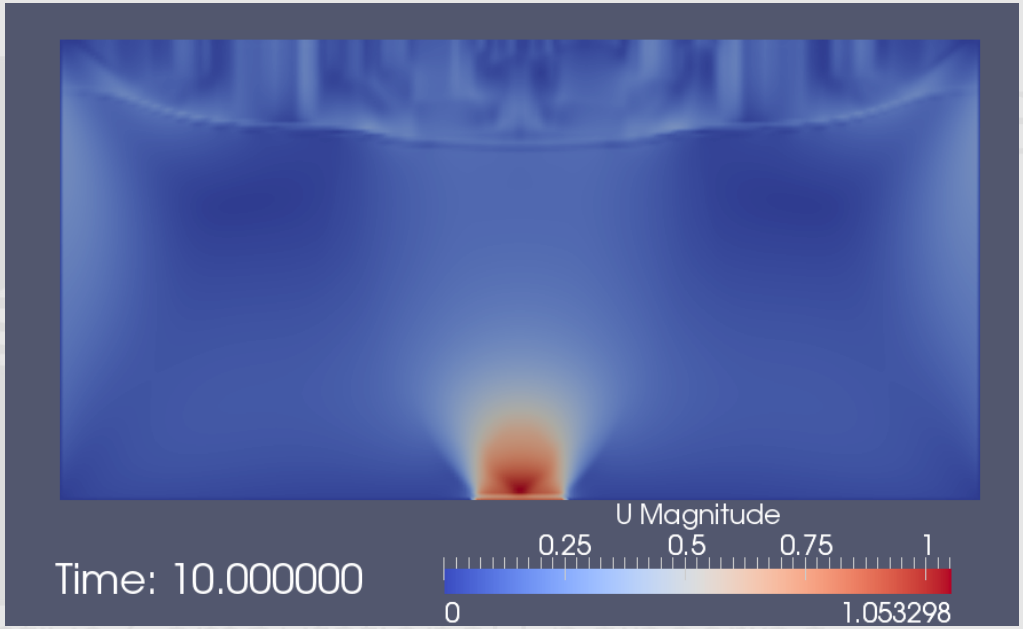


Figure : Velocity field in the whole domain

One more velocity

- Currently we're only interested in the movement of the sand phase
  - Only if there is sand
- According to the case-setup the maximum `alpha1` is 0.6
- We want a new field `USand` for postprocessing
  - Field is `U1` if `alpha1` is > 0.58
  - 0 elsewhere
- There is a utility to quickly generate such a field ...

funkySetFields

- `funkySetFields` is a utility to create or modify OpenFOAM-fields using only expressions
  - It is the oldest part of the swak-bunch
- `FSF` always needs the time for which it should be used to be specified on the command line
- For the further information two modes are possible:
  1. `expression` and all the other parameters are specified on the command line
     - This is nice for quickly modifying fields and trying out stuff
     - Only one expression can be entered at a time
  2. Additional parameters are specified in a dictionary file
     - One entry `expressions` with a list of dictionaries is needed
     - Basic parameters have the same names as the command line options

Common options of `funkySetFields`

-create Create a new field

-field name of the field-file

-expression A string with the actual expression to be evaluated. The type of this expression determines **where** the result is defined (`vol`, `surface`, `point`) and what the base type is (`scalar`, `vector`, `tensor`, ...). Without `-create` the type of the result has to fit the existing field

-condition if an existing field is manipulated (no `-create`) then only cells/faces where this logical expression evaluates to `true` will be set with the result of `expression`

-keepPatches Usually on all appropriate patches the boundary condition `zeroGradient` is set. This tells `FSF` to keep the previous boundary conditions

Calculating `USand`

- The new velocity is calculated with

```
1 > funkySetFields -latestTime -create -field <brk>
    <cont>USand -expression "alpha1>=0.58 ? U1 <brk>
    <cont>: vector(0,0,0)"
```

`alpha1, U1` any field present at the time-step in question can be used by name

`vector(0,0,0)` This initializes a volume vector field with all components 0

`>=` Logical operator greater or equal

0.58 numeric constants are interpreted as a volume scalar field

`? :` This is a common C++/C/Java operator. `a ? b : c` means "if `a` is `true` use the value `b` else use `c`"

Sand velocity without air

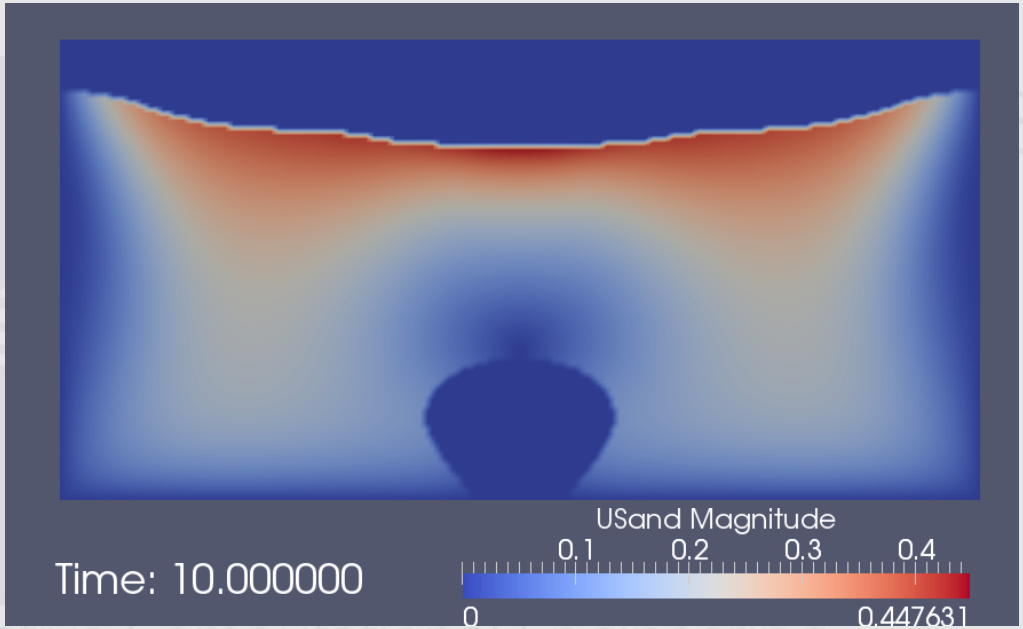


Figure : Velocity field in regions with much sand

Example: Checking the definition of `Ur`

This and the following examples can be tried out (there are no demo pictures)

```
> funkySetFields -latestTime -create -field <brk>
    <cont>UrCheck -expression "Ur-(U1-U2)"
```

`+ - * /` The regular arithmetic operation

Example: normalizing the velocity

```
1 > funkySetFields -latestTime -create -field <brk>
    <cont>UNorm -expression "U/max(mag(U))"
```

mag Function that returns the absolute value of the argument

max Returns a volume field with the maximum value of the argument

Example: Divergence of `phi`

```
1 > funkySetFields -latestTime -create -field <brk>
    <cont>divPhi -expression "div(phi)"
```

div Divergence of the arguments. The other differential operators are defined as well

Example: how far does air travel during the timestep

```
> funkySetFields -latestTime -create -field <brk>
    <cont>moveAir -expression "deltaT()*U2"
```

deltaT Returns the time-step size

time Current simulation time (not used here))

Example: distance from the highest pressure

```
1 > funkySetFields -latestTime -create -field <brk>
    <cont>pDist -expression "pos()-maxPosition(<brk>
    <cont>p)"
```

pos Function that returns a vector with the cell-centers (current position)

maxPosition Returns a volume field with the **position** of the maximum of a field

Example: normalized x-position

```
1 > funkySetFields -latestTime -create -field <brk>
    <cont>normX -expression "pos().x-<brk>
    <cont>interpolateToCell(min(pts().x))/<brk>
    <cont>interpolateToCell(max(pts().x)-min(<brk>
    <cont>pts().x))"
```

.x Gets the x-component from a vector value (of course works for other components too. .xx etc for tensor components)

pts returns a **point** field with the positions of the vertexes

interpolateToCell Because combining volume and point fields is not possible this function interpolates the point values back to cells

- **Note:** swak4Foam never interpolates implicitly. Even in situations where it might seem "logical"

Example: difference between two timesteps

```
1 > funkySetFields -time 10 -create -field <brk>
    <cont>dAlpha1 -otherCase . -otherTime 9 -<brk>
    <cont>expression "alpha1-other(alpha1)"
```

-otherCase Declare a "partner"-case. swak4Foam will automatically interpolate values from that case onto the mesh of the current case. Here we use the same case

-otherTime Time to be used for the other case

other This "function" interpolates from the other case and time

Hands On

- Try out the following:
  - Velocity of the air if `alpha1` is smaller than 0.05
  - Distance from the biggest change in `Ur`
- Or whatever else you can think of

Outline

What comes next

- Until now the case doesn't resemble the real pit at all
- Changes to be done
  - Set inhomogeneous initial conditions
    - Sand surface with a slight bump
  - Set the velocity at the bottom to go down
    - With a parabolic profile
    - Only start after a certain time
- In addition do some evaluations during the simulation
  - Velocity of the sand phase
  - Total amount of sand
  - Extreme values of certain variables

Getting files for the case with groovyBC

- Now we get the files of the case with groovyBC
- First go to the right directory

```
cd $HOME/swak4FoamTraining
```

- Extract the case files

```
> tar xvzf /home/openfoam/training_materials/<brk>
  <cont>Track3-3/sandPit.groovBCetc.tgz
```

- Go to the case directory

```
cd sandPit.groovBCetc
```

- Prepare

```
blockMesh
cp 0/alpha1Start 0/alpha1
```

Dictionary more for `funkySetFields`

- This mode is "triggered" when `FSF` is called without the `-expression`-option
- Automatically uses the file `system/funkySetFieldsDict`
  - different file can be used with `-dictExt`
- Dictionary file has an entry `expressions` with a list of dictionaries
  - Each dictionary acts like a command-line call to `FSF`
    - Dictionary entries corresponding to command line options
- A `-time`-option has to be specified anyway

```
funkySetFields -time 0
```

The `variables`-list

- Sometimes expressions get very complex
  - Splitting them up into multiple expressions improves readability and performance
- All swak4Foam-parsers support an entry `variables` with a list of variable declarations
  - Evaluated before the actual expression and variables can be used in the expression (and other variable declarations)
- Format of a variable declaration is `name=expr;` with

  name a valid name

  expr an expression valid for the current parsers

system/funkySetFieldsDict for the sand-pit

```
expressions
(
        init
        {
                field alpha1;
                expression "0.6";
                condition "pos().z<min(idealHeight,levelHeight)";
                keepPatches true;
                variables (
                   "minX=interpolateToCell(min(pts().x));"
                   "maxX=interpolateToCell(max(pts().x));"
                   "minZ=interpolateToCell(min(pts().z));"
                   "maxZ=interpolateToCell(max(pts().z));"
                   "levelHeight=0.75*(maxZ-minZ)+minZ;"
                   "lowerHeight=levelHeight-0.25*(maxZ-minZ);"
                   "width=min(2*(levelHeight-lowerHeight),maxX-minX)<brk>
                      <cont>;"
                   "middle=0.5*(minX+maxX);"
                   "idealHeight=pow((pos().x-middle)/(width*0.5),2)<brk>
                      <cont>*(levelHeight-lowerHeight)+lowerHeight;"
                );
        }
);
```

Distribution of `alpha1` at $t = 0$

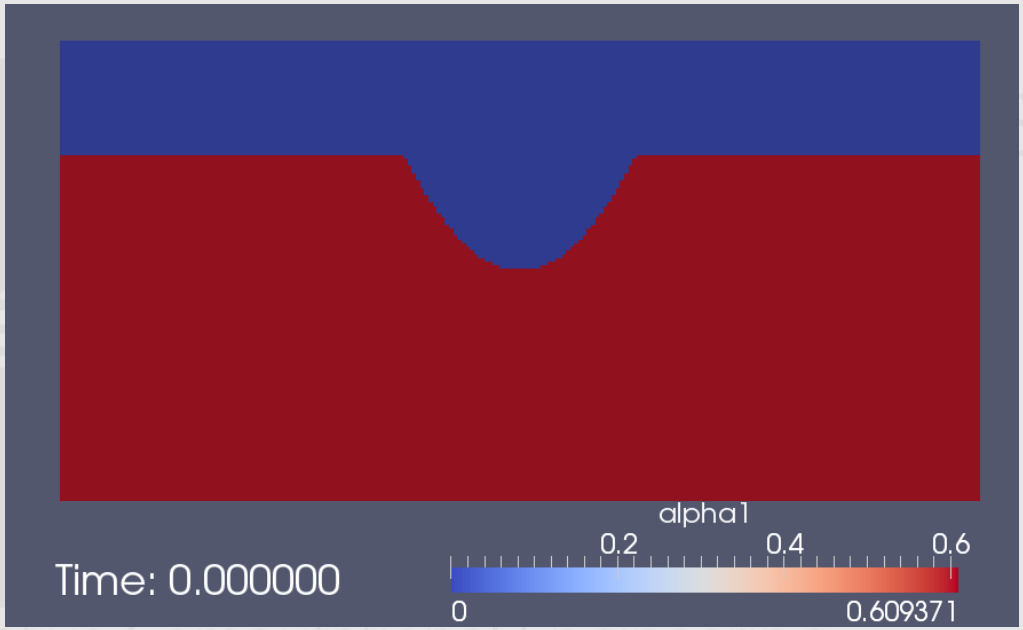


Figure : Initial distribution of `alpha1`

Excurse: available parsers

Until now we always used the `internalField`-parser. Available parsers in swak4Foam are

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`
`internalFaField`	Internal values of a FAM-field (`1.6-ext` only)
`faPatch`	Boundary patch of a FAM-field (`1.6-ext` only)

Preparing case for groovyBC

- groovyBC is not a standard part of OpenFOAM
  - So it has to be added to the solver
  - Fortunately OpenFOAM supports a plugin-mechanism
- "Plugins" are loaded when the solver starts up via the libs-entry in the controlDict
  - Then it can be used like any other boundary condition

system/controlDict

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

Fields of groovyBC

- groovyBC allows setting a mixture of Dirichlet and Neuman boundary conditions
  - Default is Dirichlet
- Entries are

  valueExpression Expression with the value to be set

  gradientExpression Optional expression with the gradient to be set

  fractionExpression Optional expression that determines whether to use the value (1) or the gradient (0) or something in between
- Additional entries like `variables` from the swak4Foam-parser

The sand velocity

Part of U1

```
bottom
{
    type                groovyBC;
    value               uniform (0 0 0);
    variables (
        "minX=min(pts().x);"
        "maxX=max(pts().x);"
        "middle=0.5*(minX+maxX);"
        "vMax=1;"
        "profile=vMax*(1-pow((pos().x-middle)/(middle-minX)<brk>
            <cont>,2));"
        "tOpen=5;"
        "tOpening=1;"
        "factor=time()<tOpen ? 0 : (time()>(tOpen+tOpening)<brk>
            <cont> ? 1 : (time()-tOpen)/tOpening);"
    );
    valueExpression "normal()*factor*profile";
}
```

Functions in the sand velocity

normal() Vector normal to the current faces (this makes the expression independent from the orientation of the patch)

time() The current time in the calculation

pos() Position of the face

pts() Positions of the **vertexes** of the faces

The air velocity

The same as the sand, but a bit higher

- `value` is only needed during startup (set by the expression later)

Part of `U2`

```
bottom
{
    type                groovyBC;
    value               uniform (0 0 0);
    valueExpression     "1.1*U1";
}
```

Velocity set by groovyBC

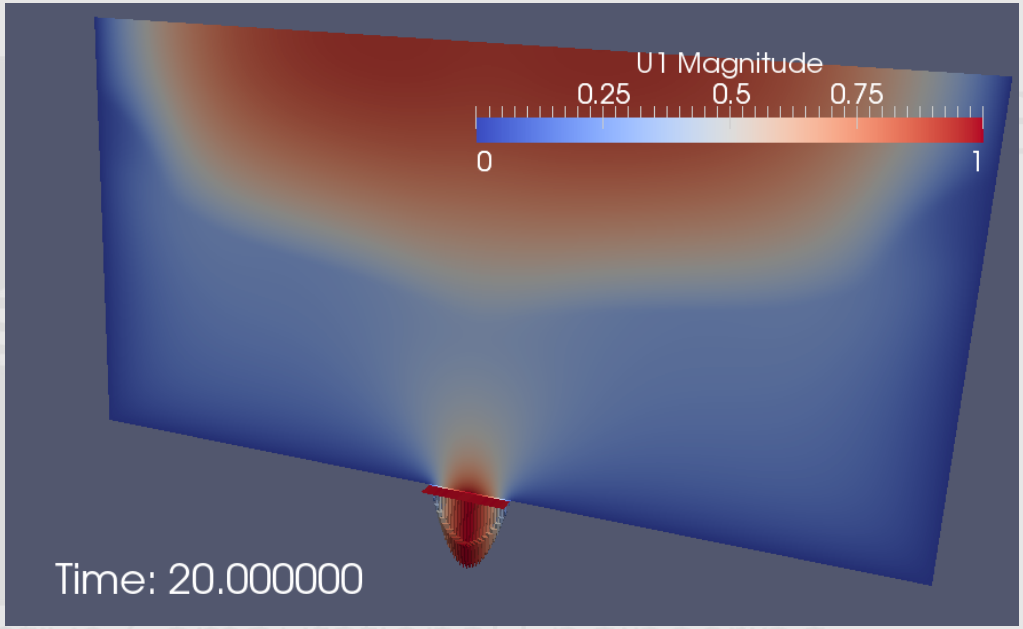


Figure : Parabolic outlet velocity

Velocity set by groovyBC - closeup

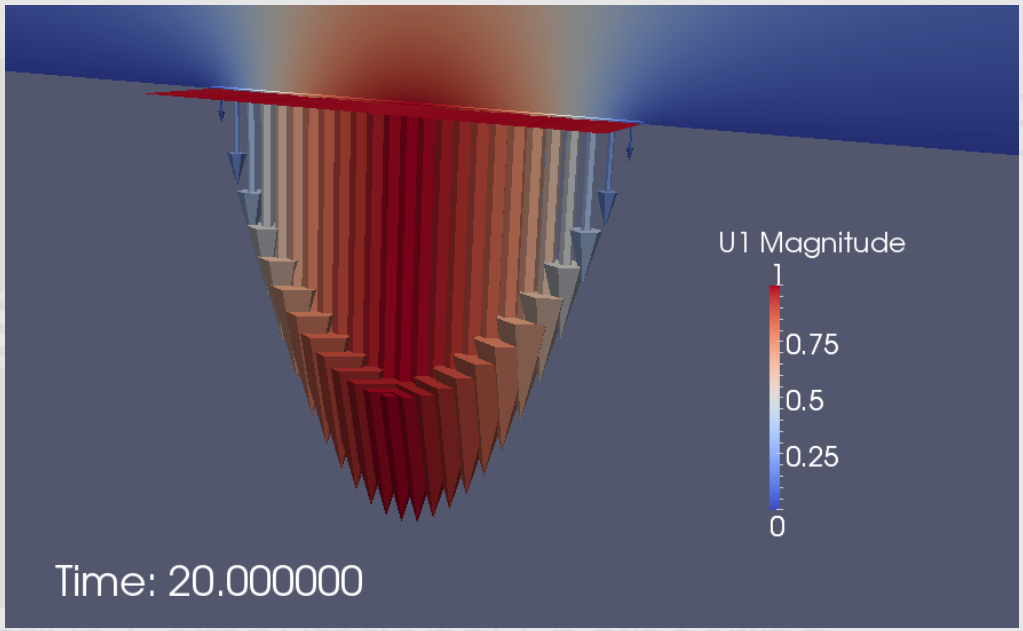


Figure : Closer look at the velocity

Preparing and running the case

Before running the case we prepare it using a script

prepare.sh

```
pyFoamClearCase.py .
rm -f 0/alpha1 0/alpha1.gz

blockMesh

cp 0/alpha1Start 0/alpha1

funkySetFields -time 0
```

Distribution of `alpha1` at $t = 5.3$

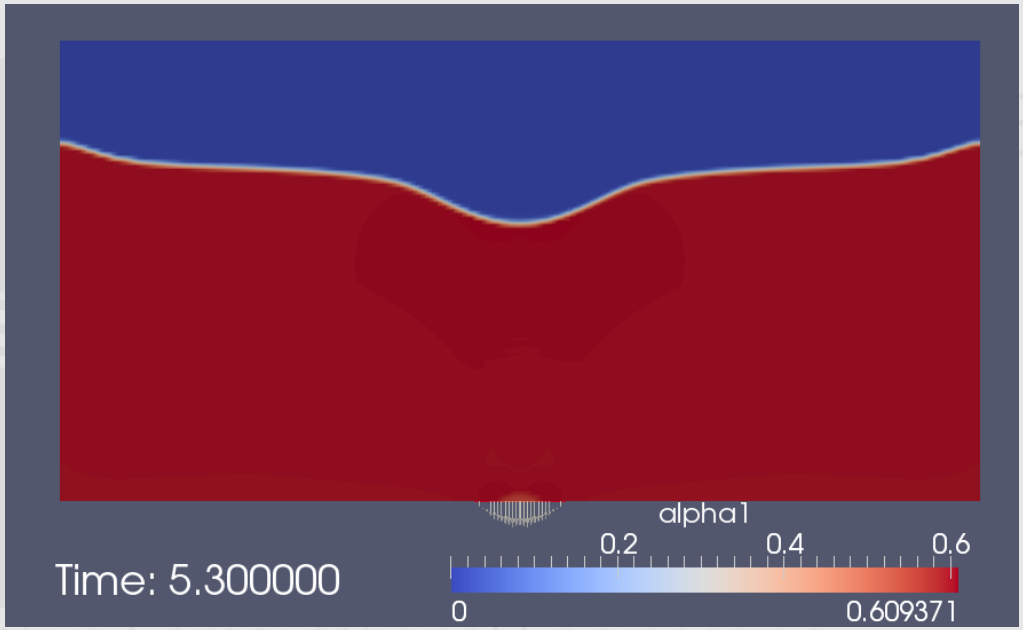


Figure : Distribution of `alpha1` as the pit opens

Distribution of `alpha1` at $t = 50$

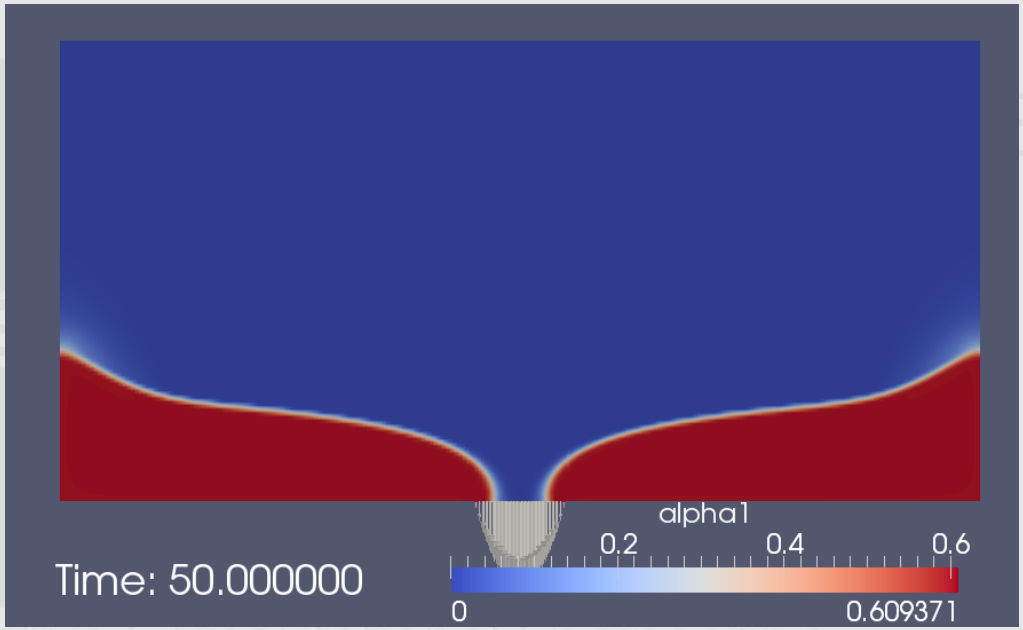


Figure : Final distribution of `alpha1`

Function objects

- function-objects are little programs that are evaluated at the end of each time-step
  - Usually used for post-processing
- Most of the functionality of swak4Foam are functionObjects
  - In the following examples it will only be described what it does but not every keyword/parameter
- Function objects are defined in a dictionary `functions` in `controlDict`

Adding plugins for swak-function-objects

- `simpleFunctionObjects` are function objects that don't rely on expressions
  - Write data at each timestep
  - `simpleSwakFunctionObjects` are based on these but evaluate expressions

system/controlDict

```
libs (
    "libgroovyBC.so"
    "libsimpleFunctionObjects.so"
    "libsimpleSwakFunctionObjects.so"
);
```

Extremes of fields

SimpleFunctionObjects

- This writes the minimum and the maximum of `fields` to files
- `verbose` makes sure that the output gets also printed to the screen
- Evaluates at every timestep
  - New option `outputControlMode` allows changing that

`functions` in `controlDict`

```
1 extremes  {
    type volumeMinMax;
3   verbose true;
    fields (
5             U1
              U2
7             alpha1
              p
9   );
}
```

Writing the velocity of sand

swakExpression

- Evaluates an expression
  - Based on simpleFunctionobjects
- valueType determines the parser
- accumulations reduces the field to a single value

Entry in functions

```
sandVelocity {
    type swakExpression;
    valueType internalField<brk>
        <cont>;
    verbose true;
    variables (
        "thres=0.55;"
    );
    expression "alpha1><brk>
        <cont>thres ? mag(U1<brk>
        <cont>) : 0";
    accumulations (
        max
        weightedAverage
    );
}
```

Velocity of sand during simulation

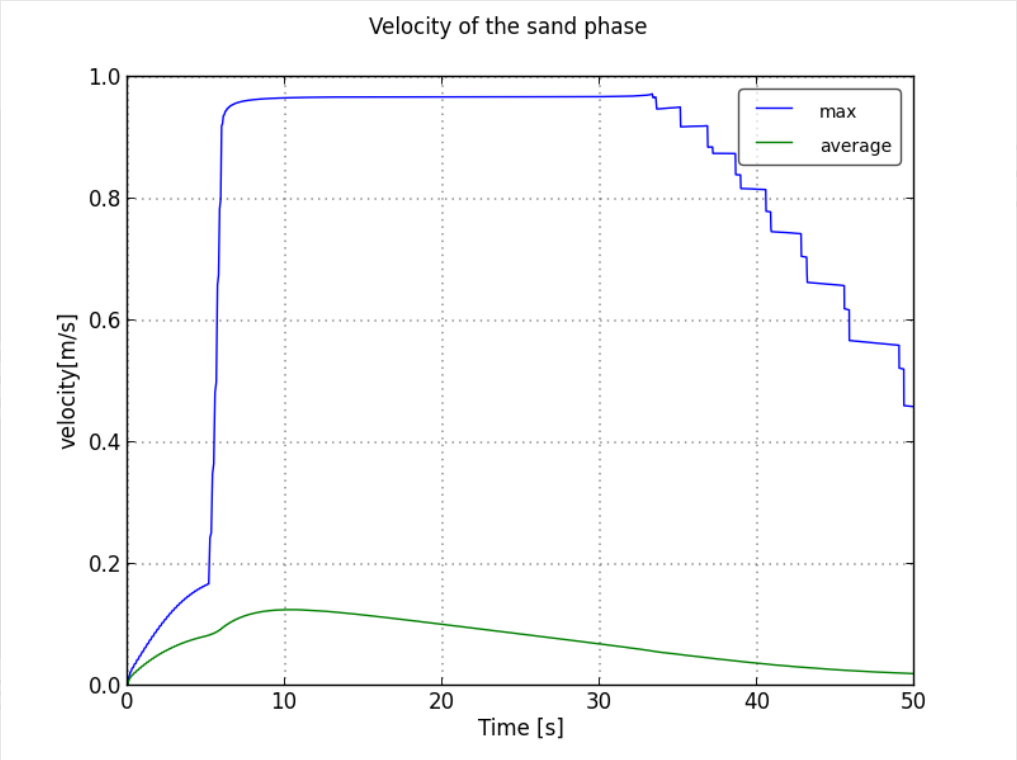


Figure : How much sand is there

Getting total amount of sand

One more `swakExpression`

- "Inherits" most settings from previous function object
- `vol()` returns the volume of a cell

Adding to `controlDict`

```
totalSand {
    $sandVelocity;
    expression "vol<brk>
        <cont>()*<brk>
        <cont>alpha1"<brk>
        <cont>;
    accumulations (
        sum
    );
}
```

Amount of sand in the simulation

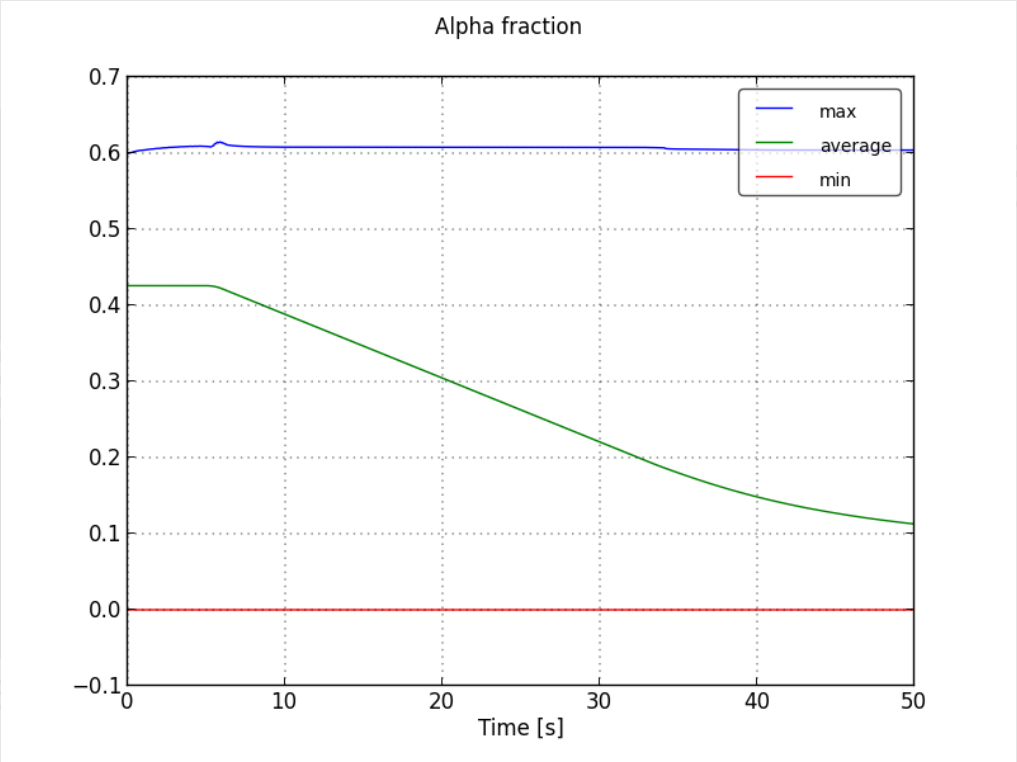


Figure : How much sand is there

Hands On

- Try the following
  - Modify the boundary condition of the velocity to pulsate (`sin`) with `time`
  - Instead of `zeroGradients` fix `alpha1` to a value that is a function of y
- Or whatever other craziness you can think of

Outline

What happens here

- We add a cloud of particles (the oranges) to the evaluation
  - To make this work we have got to add a new viscosity field
- Also fields with the Courant-number are evaluated

Getting files for the case with particles

- Now we get the files of the case with added particles
- First go to the right directory

```
cd $HOME/swak4FoamTraining
```

- Extract the case files

```
1 > tar xvzf /home/openfoam/training_materials/<brk>
    <cont>Track3-3/sandPit.withParticles.tgz
```

- Go to the case directory

```
cd sandPit.withParticles
```

It seems that the particles don't behave correctly with the version 2.2.0 of OpenFOAM installed on the stick. The following pictures are taken from a calculation with 2.2.x

Adding new extensions

swakFunctionObjects function objects based on swak-expression that do **not** write timeline data

swakVelocityFunctionPlugin Extends the parser with velocity-related functions

simpleLagrangianFunctionObjects Function objects that add simple clouds

```
libs (
    "libgroovyBC.so"
    "libsimpleLagrangianFunctionObjects.so"
    "libsimpleFunctionObjects.so"
    "libsimpleSwakFunctionObjects.so"
    "libswakFunctionObjects.so"
    "libswakVelocityFunctionPlugin.so"
);
```

Function plugins

- This allows adding specialized functions to the parsers
  - Without modifying them
  - Just add the library in `libs`
    - The parser knows about them and will print information about the available functions the first time he is called
    - That way they can also be used from `funkySetFields`
- Application:
  - Functions that are not needed by everyone
  - Evaluations that can not be written as a simple expression
- A number of plugin-libraries come with swak4Foam:
  - Related to turbulence, thermophysical models, discretizations etc
- Anyone can write plugin-functions for his special needs

Writing the local Courant-numbers

expressionField

- Creates a new field `fieldName` from `expression`
- `autowrite` determines whether it will be written
- `courantIncompressible` is plugin function
  - Needs a `surfaceScalarField` as the argument

In functions

```
airCourant {
    type expressionField;
    autowrite true;
    fieldName courantU1;
    expression "<brk>
        <cont>courantIncompressible<brk>
        <cont>(phi1)";
}
sandCourant {
    type expressionField;
    autowrite true;
    fieldName courantU2;
    expression "<brk>
        <cont>courantIncompressible<brk>
        <cont>(phi2)";
}
```

Viscosity for the particles

- The particle cloud needs a viscosity
  - Calculated as a mixture of the phases
  - "thick" enough to make the particles "lie" on the sand surface

```
totalMu {
    type expressionField;
    expression "mu1+nut2*rho";
    fieldName totalMu;
    autowrite true;
}
```

Adding a particle cloud

- Needs some fields
  - Can be "regular" fields or newly calculated ones
- Actual properties of the cloud are in a separate file `<cloudName>Properties` (here `tracerBallsProperties`)
  - This is beyond the scope of this presentation

```
balls {
    type evolveKinematicCloud;
    cloudName tracerBalls;
    rhoName rho;
    UName U;
    muName totalMu;
}
```

Particles coming in

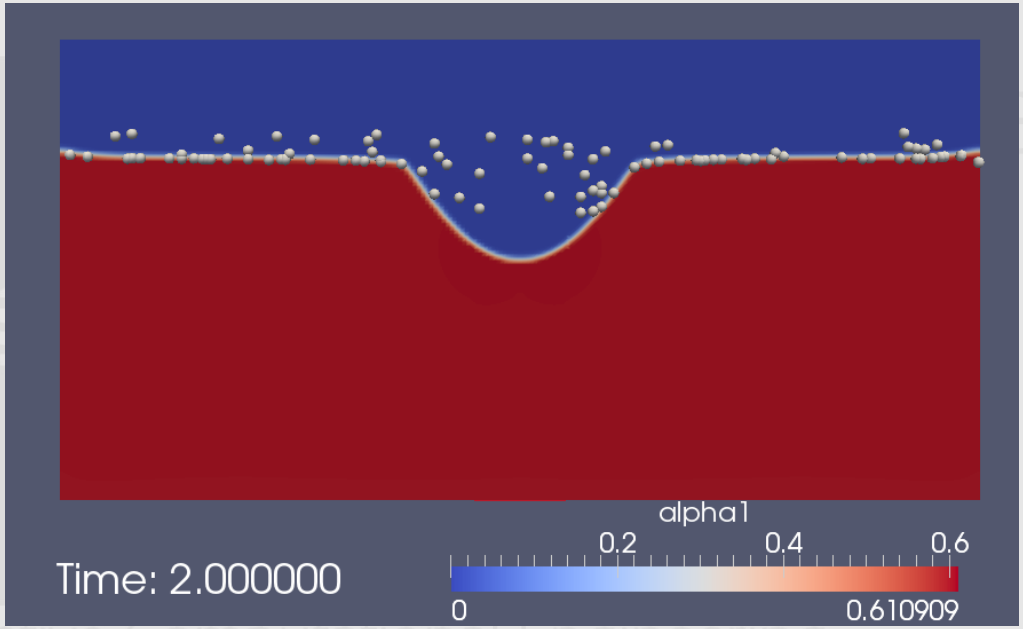


Figure : Particles falling from the sky

Particles lying on surface

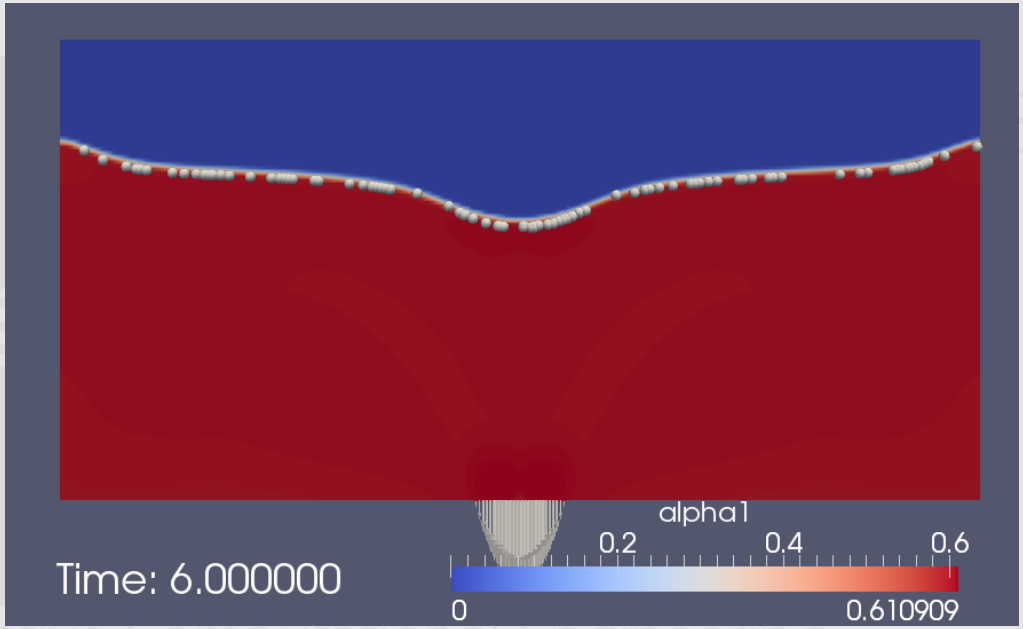


Figure : Moving with the surface

Plotting cloud quantities

- Cloud writes statistics to the screen (this is standard-OpenFOAM)
  - Processed by PyFoam

customRegexp

```
kineticEnergy {
                expr "Linear kinetic energy           = (%f%)";
                type slave;
                master NumberOfParticles;
                titles (energy);
}
NumberOfParticles {
                    expr "Current number of parcels        = (%f%)";
                    theTitle "Particles and their movement";
                    titles ( nr );
                    xlabel "t[s]";
                    y2label "nr[1]";
                    ylabel "energy";
                    alternateAxis ( nr );
}
```

Particle number and energy

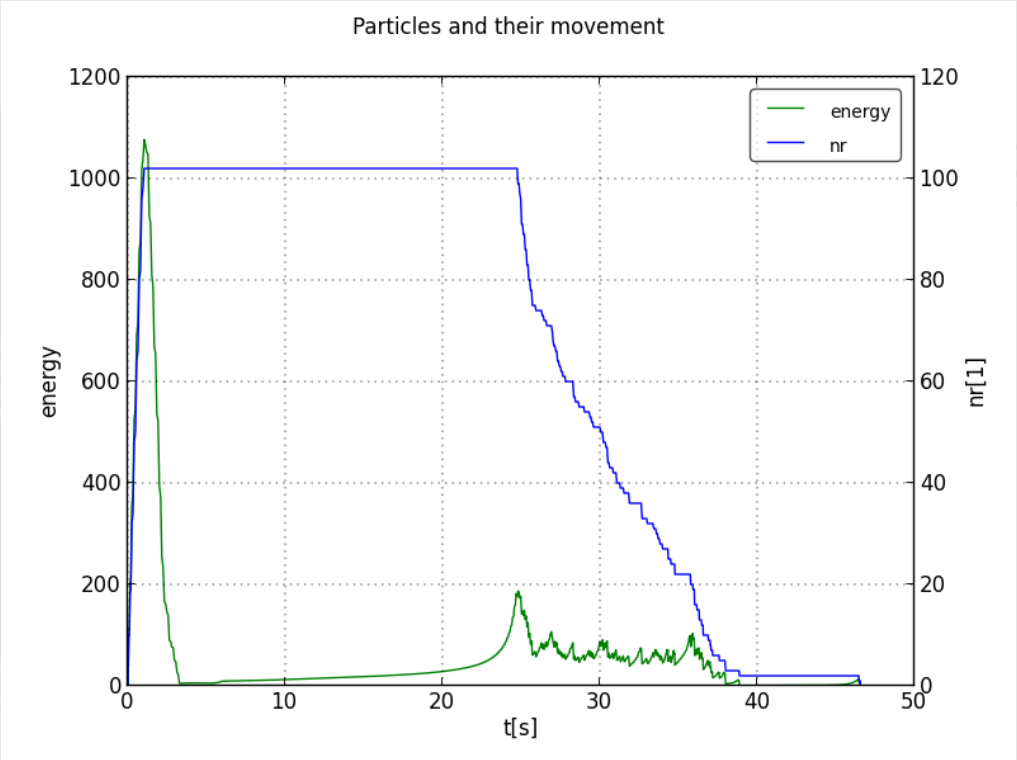


Figure : Number and kinetic energy during simulation

Outline

Outline

Biggs Darklighter

- In A New Hope (also known as "the first Star Wars movie" but **never** as Episode IV) the childhood friend Biggs Darklighter of Luke Skywalker is introduced
  - He dies during the attack on the Death Star
- Since then rumors say that the reason for his death was a damage to his nervous system do to chemical intoxication and the slower reflexes caused by that
  - Intoxication was due to inhaling the exhaust gas from Lukes landspeeder when racing through the canyons of Tatooin ... it is said

The original landspeeder

Figure : The original landspeeder (Source: www.revell.com)

Landspeeders in a canyon

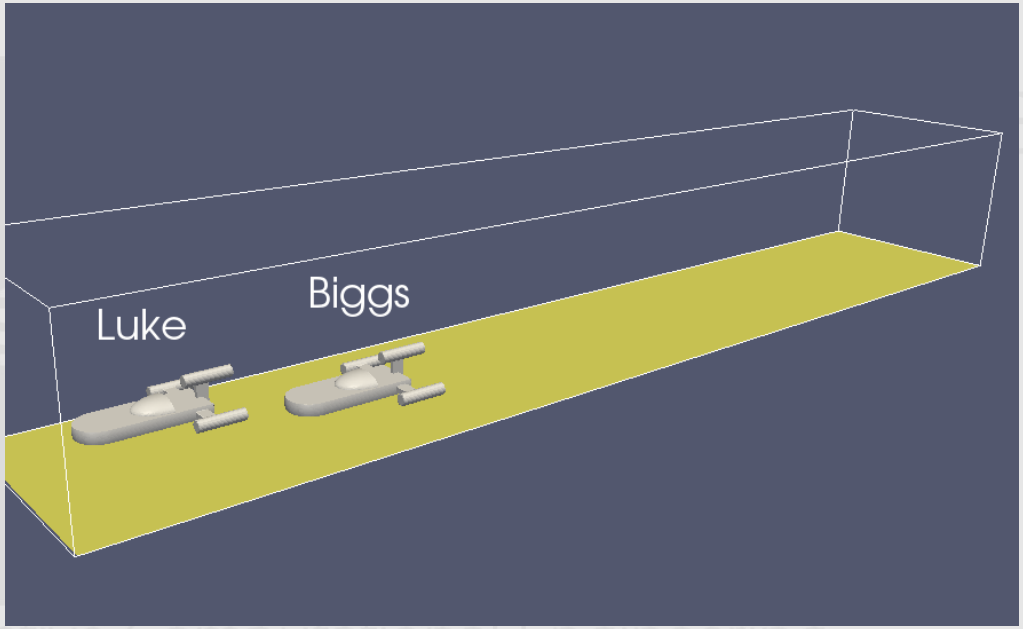


Figure : The two landspeeders in the canyon

Outline

Getting files for the landspeeder-case

- Now we get the files for the race in the canyon
- First go to the right directory

```
cd $HOME/swak4FoamTraining
```

- Extract the case files

```
1 > tar xvzf /home/openfoam/training_materials/<brk>
    <cont>Track3-3/landSpeedersInCanyon.tgz
```

- Go to the case directory

```
cd landSpeedersInCanyon
```

Adding new searchableSurfaces

- swak4Foam has a number of new surfaces that can be used with snappyHexMesh
  - They are added by loading the library

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

- This adds
  - Boolean operations (combining surfaces)
  - Additional primitives
  - Coordinate transformations of existing surfaces
- All this works of course for STLs too
  - To mix and match them

Definition of boolean operations

- Define a `type`
  - `union`, `intersection`, `difference` and `exclusive`
- Define the two surfaces `a` and `b` that are combined
  - Only for `difference` the results are different if they are exchanged

Boolean operations

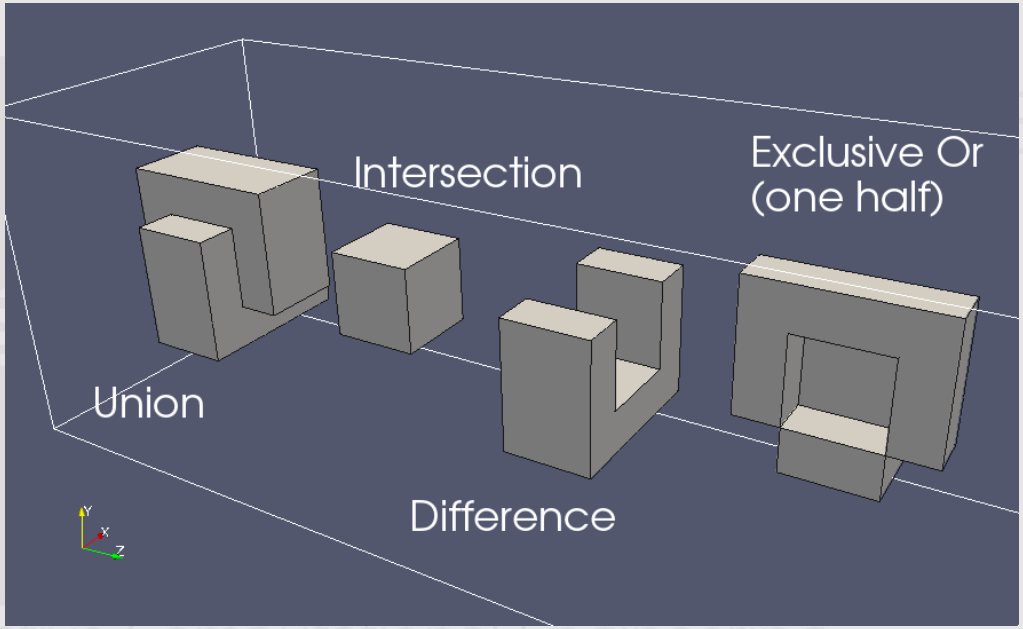
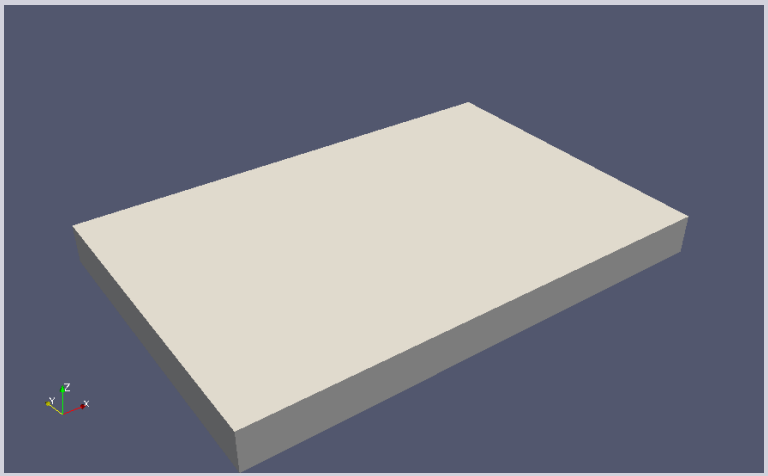


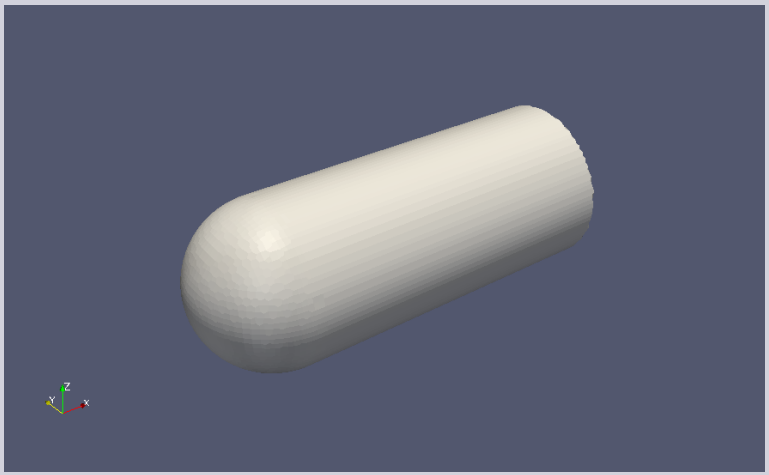
Figure : Available boolean operations

Basic blocks for the speeder

A block



Cylinder and sphere



Implementation of the body

```
speederBody {
    type union;
    a {
        type intersect;
        a {
            type searchableBoxWithDirections;
            max (3 2 0.25);
            min (-3 -2 -0.25);
        }
        b {
            type union;
            aName front;
            a {
                type searchableSphereWithFix;
                centre (-1.5 0 0);
                radius 1;
            }
            bName side;
            b {
                type searchableCylinder;
                point1 (-1.5 0 0);
                point2 ( 2.5 0 0);
                radius 1;
            }
        }
    }
    b {
        type translate;
        surface {
            $windShield;
        }
        translation (1 0 0.2);
    }
}
```

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

The basic body (intersection)

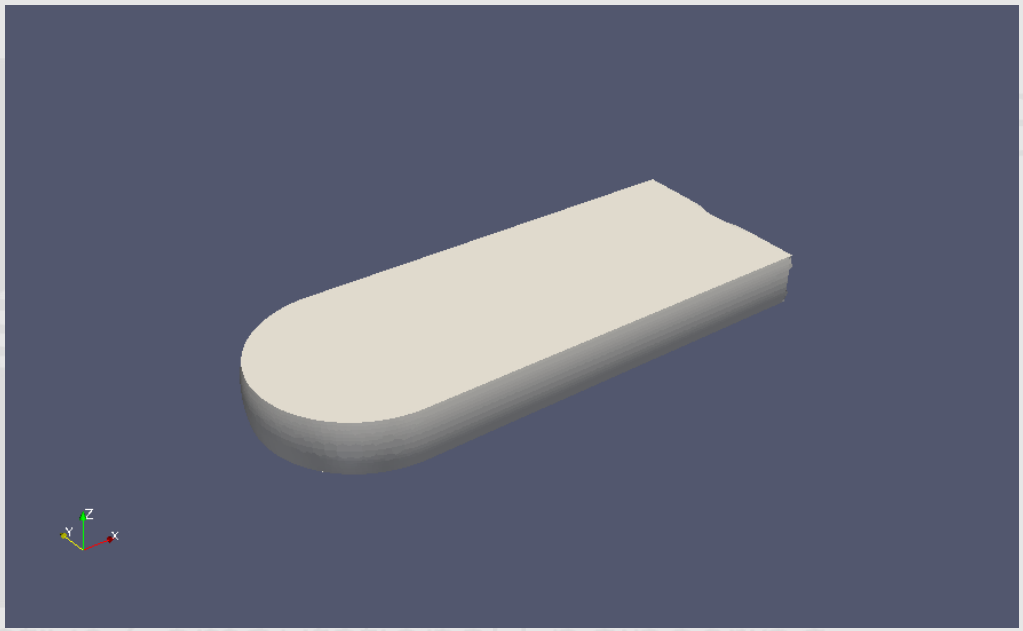


Figure : Intersection of two shapes

Jet and windshields

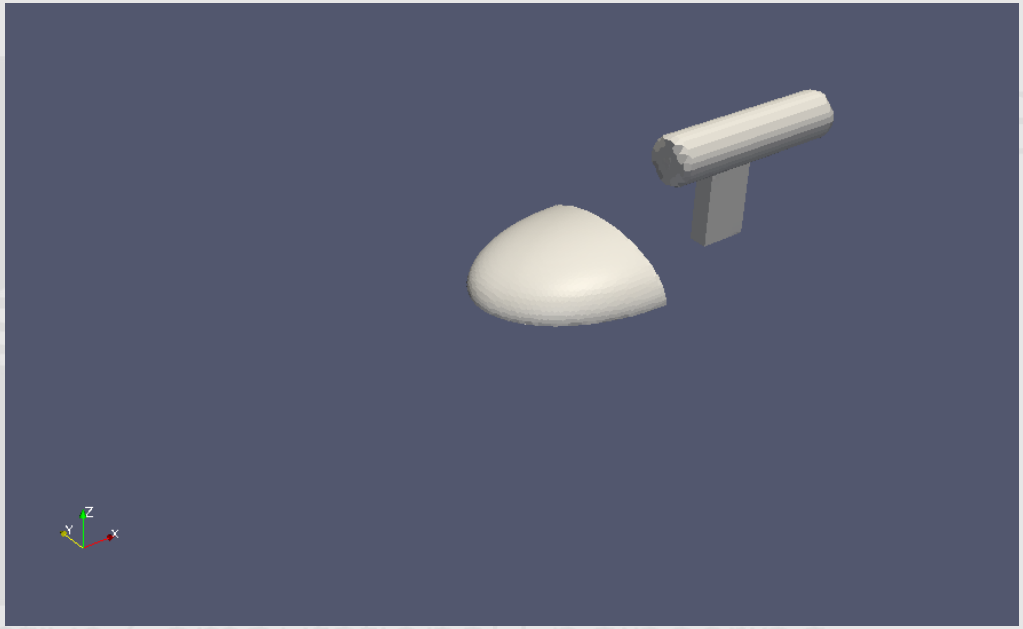


Figure : The windshield and one jet

Patch names

- Names of the patches are composed from
  - the patch names of the sub-surfaces
  - name of the operation
  - whether they belong to `a` or `b`
    - these names can be modified with `aName` and `bName`
- This generates rather long names and unnecessary patches
- Modifier-surfaces like `renameRegions` allow changing these names and `oneRegion` collects them
- But sometimes we want different parts of a surface to be different patches
  - For instance the inlet and outlet of the jets

Parts of a jet

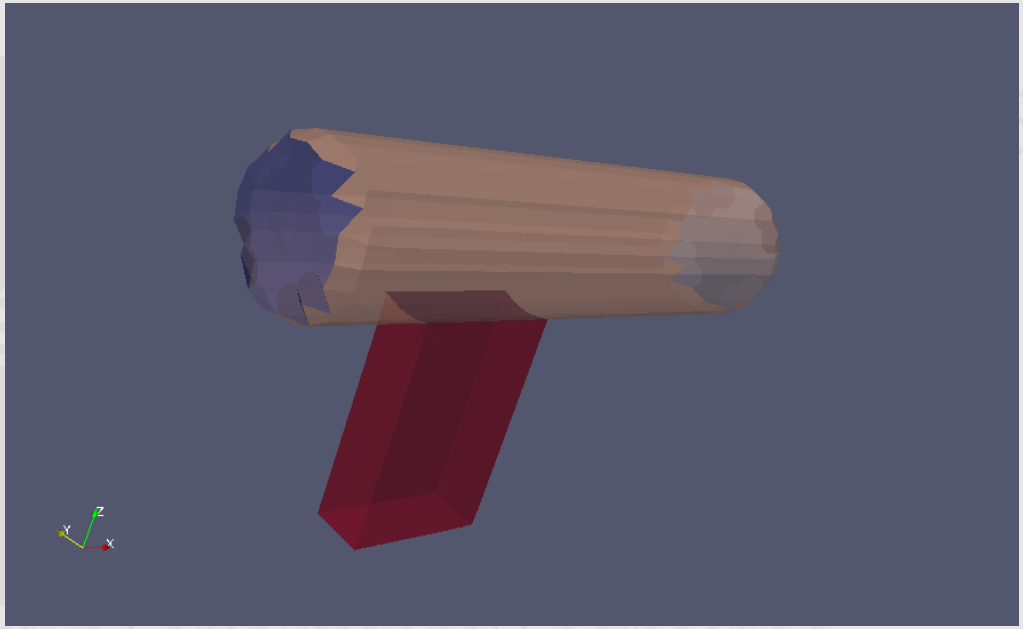


Figure : Different patches on a jet

The final landspeeder

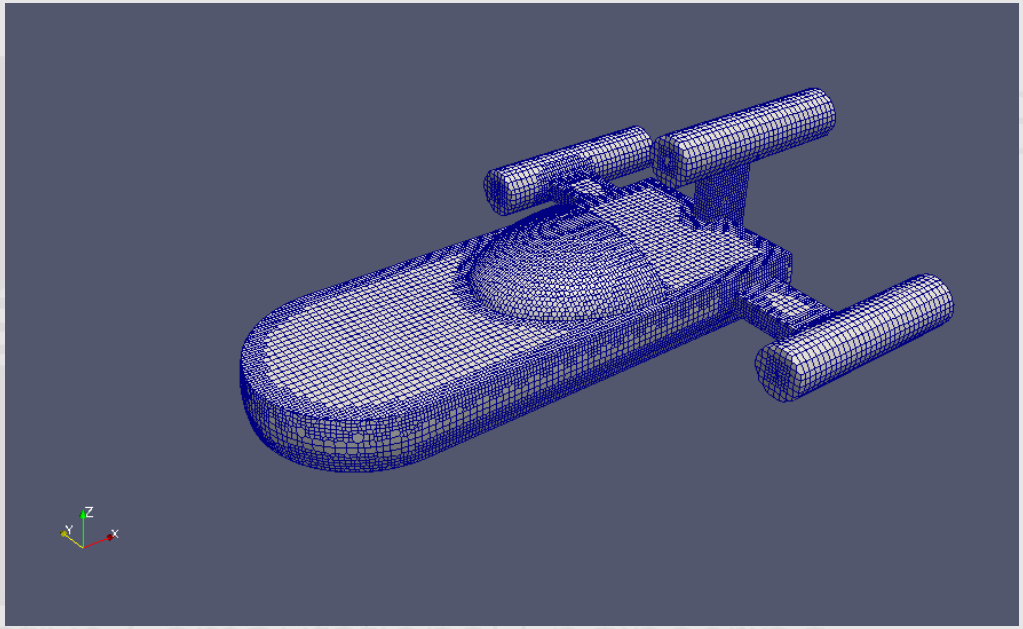


Figure : The final landspeeder with surface mesh

Outline

Solver we use

- We use `simpleFoam`
  - Case based on the `motorCylce`-tutorial
- Add a function object that adds a passive scalar for the exhaust

The prepare script

prepare.sh

```
pyFoamClearCase.sh .
rm -fr 0
rm -f constant/polyMesh/*.gz

blockMesh

snappyHexMesh -overwrite

cp -r 0.org/* 0

potentialFoam -writep -noFunctionObjects
```

Remote variables

- Entries in `variables` can be calculated on other entities
- This is indicated by an entry of the form `{valueType'name}` where `valueType` indicates which parser is used (whether it `internalField`, `patch` etc)
  - `{name}` is shorthand for `{patch'name}`
  - Entities on other meshes can be selected by adding `/regionName`
- The expression on the right side of the equal sign is evaluated with the appropriate parser
- In the general case the result must be a single value (for instance a maximum or a sum)
  - Exception are `mapped` patches

Air speed increase by the jet

U

```
"landSpeeder.+Jet_inlet"
{
    type groovyBC;
    value uniform (0 0 0);
    variables (
        "vIn{inlet}=sum(area()*U)/sum(area());"
    );
    valueExpression "0.75*vIn";
}

"landSpeeder.+Jet_outlet"
{
    type groovyBC;
    value uniform (0 0 0);
    variables (
        "vIn{inlet}=sum(area()*U)/sum(area());"
    );
    valueExpression "1.25*vIn";
}
```

Adding a passive scalar

- Since OpenFOAM 2.2 a similar function object exists in normal OpenFOAM
  - We'll use the swak-variant nevertheless
    - It allows setting the various coefficients of the transport equation with arbitrary expressions
- What needs to be added to the case
  1. The function object to `controlDict`
  2. A field-file (with the boundary conditions)
  3. Entries to `fvSolution` and `fvSchemes`
- We only show 1 and 2

The function object

controlDict

```
1  exhaustTransport {
     type solveTransportPDE;
3    solveAt timestep;
     fieldName exhaust;
5    steady true;
     diffusion "turb_nuEff()" [0 2 -1 0 0 0 0];
7    source "0" [0 0 -1 0 0 0 0];
     phi "phi" [0 3 -1 0 0 0 0];
9  }
```

Exhaust from the jets

Gas leaves the jet an exhaust concentration 1

0/exhaust

```
"landSpeeder.*_inlet"
{
    $outlet;
}

"landSpeeder.*_outlet"
{
    type fixedValue;
    value uniform 1;
}
```

Measuring the exhaust

system/controlDict

```
exhaustDistribution {
  type swakExpression;
  verbose true;
  valueType internalField;
  expression "exhaust";
  accumulations (
                 min
                 max
                 weightedAverage
                 );
}
exhaustMeasure {
  type patchExpression;
  patches (
           "landSpeeder.*_inlet"
           outlet
           );
  expression "sum(area())*exhaust/sum(area())";
  verbose true;
  accumulations ( average );
}
```

sforschung GmbH
l Engineering

Actually the measurement is not done during the first 10 steps

- Because in the beginning the passive scalar does not converge
- Special function object that wraps others
  - Executed if condition is fulfilled

Wrapped around the measurement

```
1   startLater {
      type executeIfSwakExpression;
3     readDuringConstruction true;
      valueType patch;
5     patchName inlet;
      logicalExpression "time()>=10";
7     logicalAccumulation and;
      functions {
9       exhaustDistribution {

11  ...
```

Killing diverging runs

panicDump

- function object that stops the run if a scalar field is outside the "comfort zone"
  - Writes all fields

controlDict

```
velocityMagnitude {
    type expressionField;
    fieldName magU;
    expression "mag(U)";
    autowrite false;
}
stopHighVel {
    type panicDump;
    fieldName mag(U);
    minimum -1;
    maximum 1000;
}
```

Outline

What happens here

- We run the case with

```
pyFoamPlotRunner.py --clear --progress simpleFoam
```

- run probably won't finish during this training
- Results raise some doubt in the convergence

Linear convergence

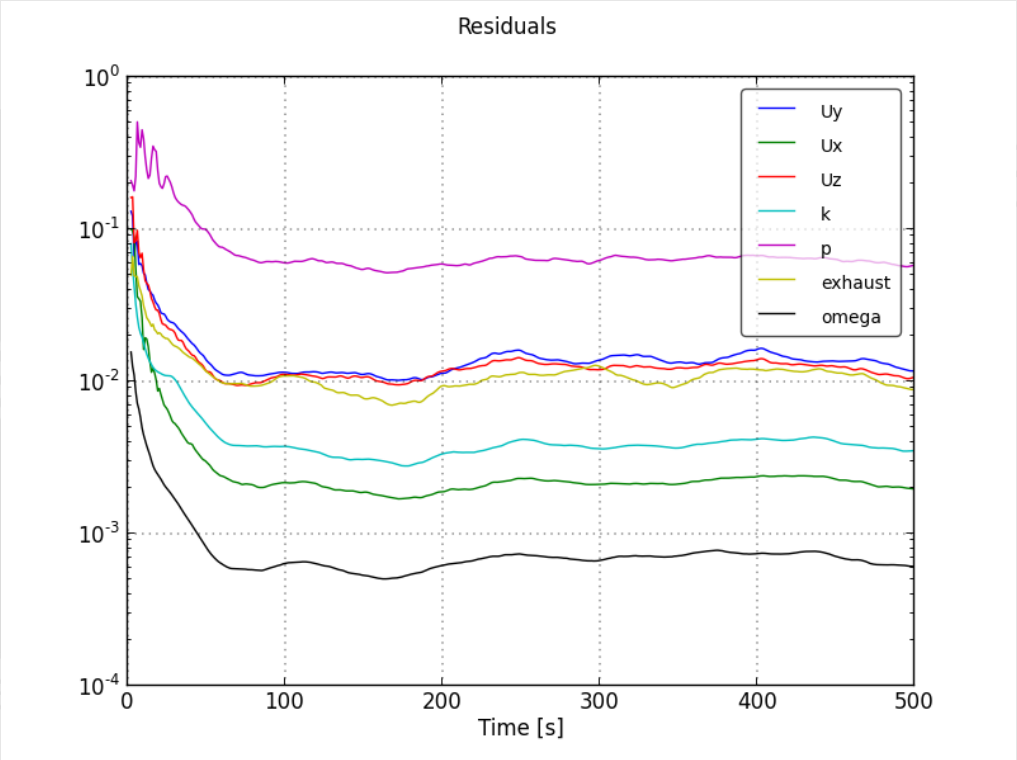


Figure : Graph of the residuals

Velocity and pressure of the landspeeders

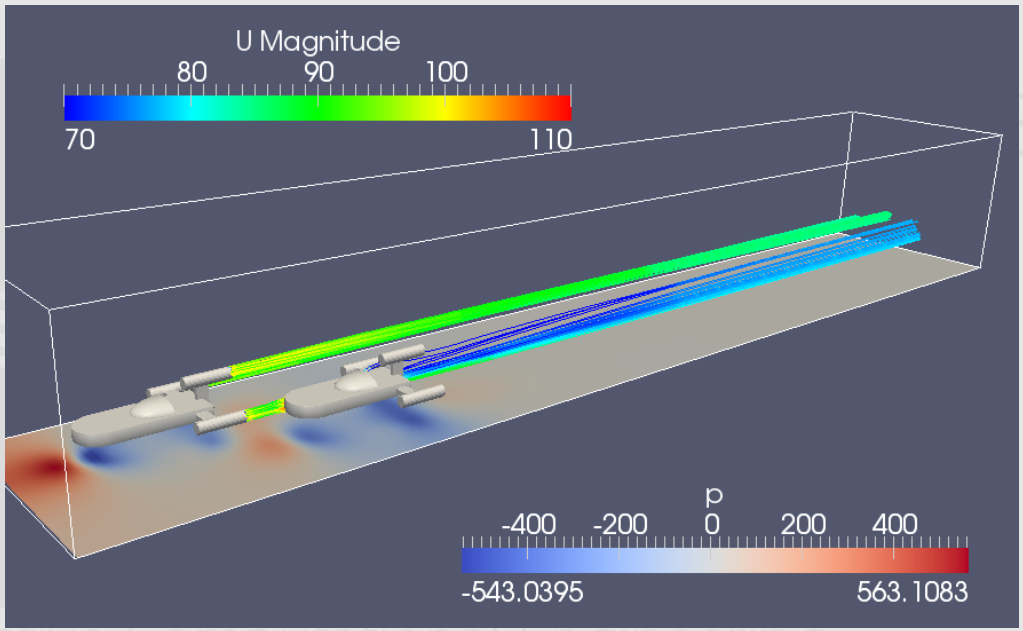


Figure : The general flow solution

Exhaust of the landspeeders

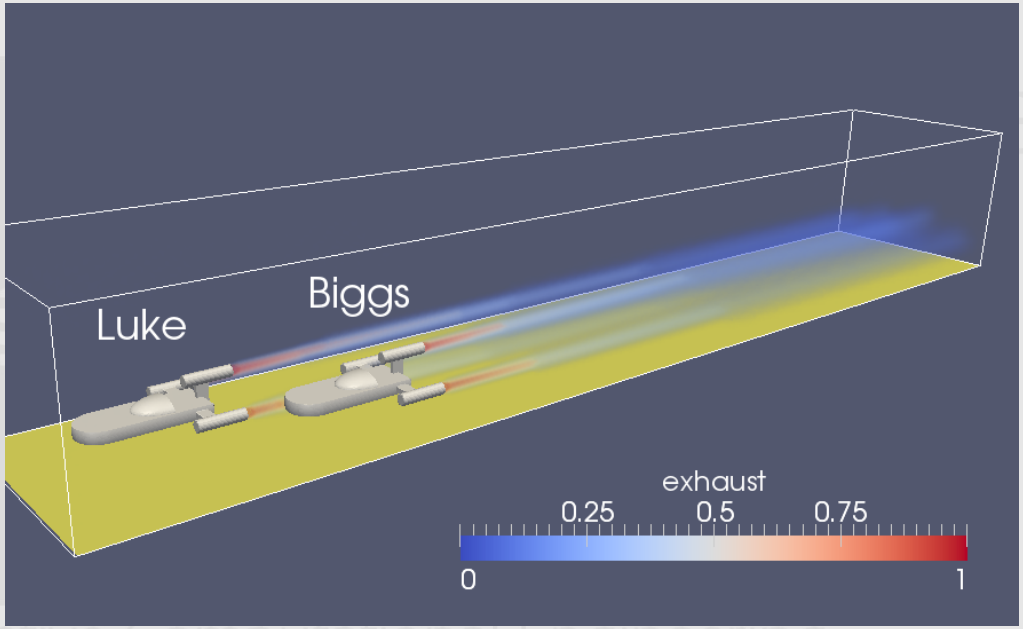


Figure : The exhaust gas of the landspeeders

Closeup view of the exhaust

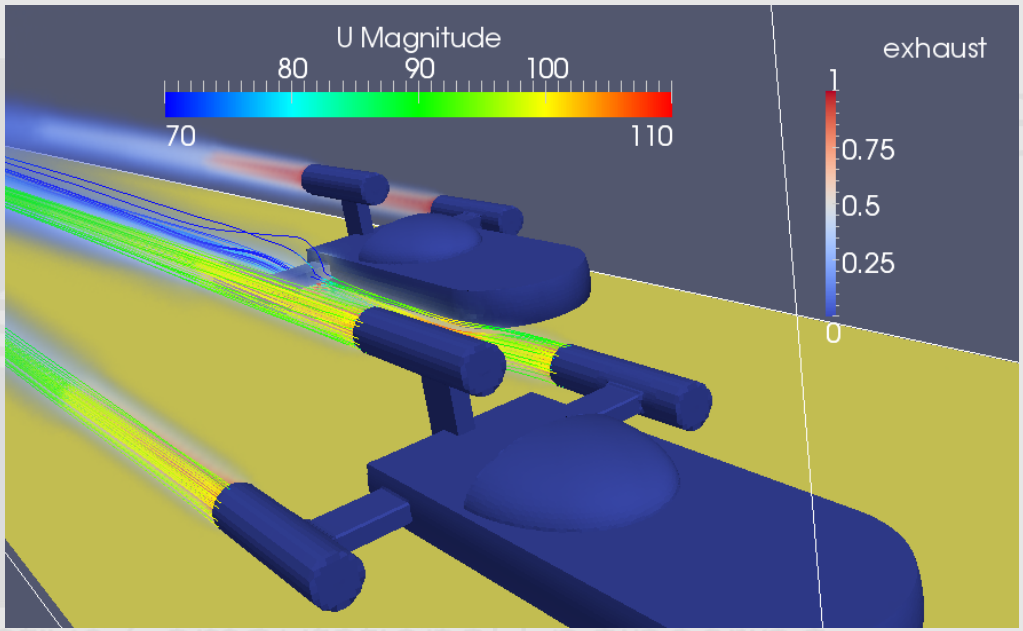


Figure : Exhaust gas around the speeders

Average exhaust concentration on patches

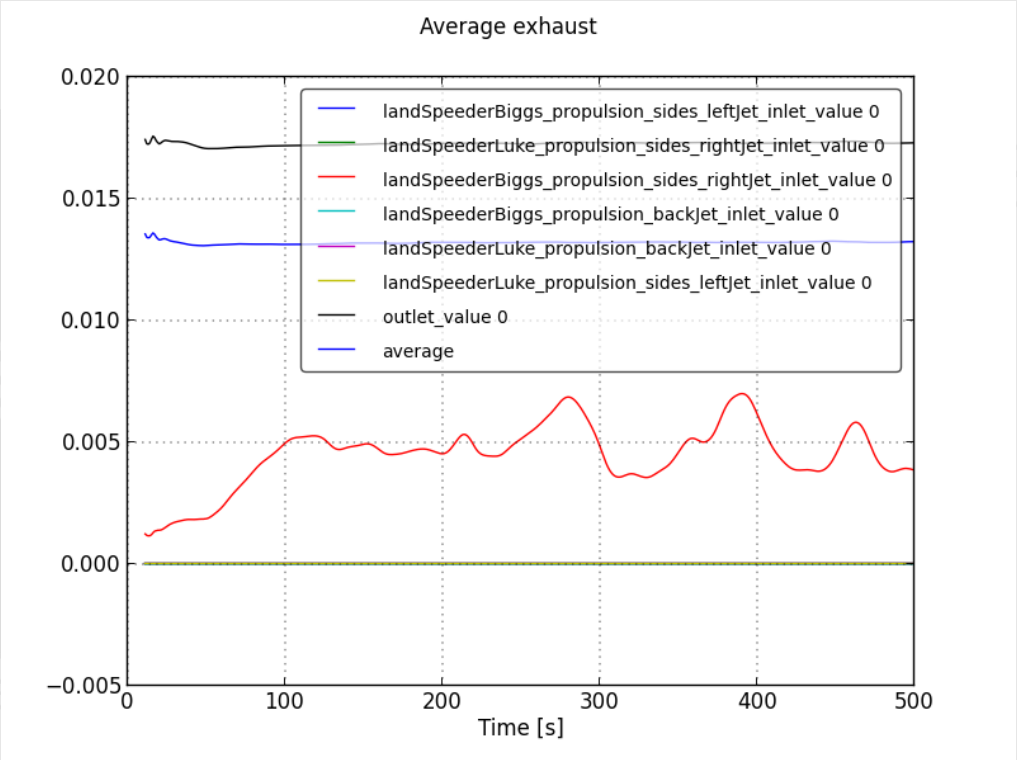


Figure : Exhaust concentrations during simulation

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Exhaust on Biggs speeder

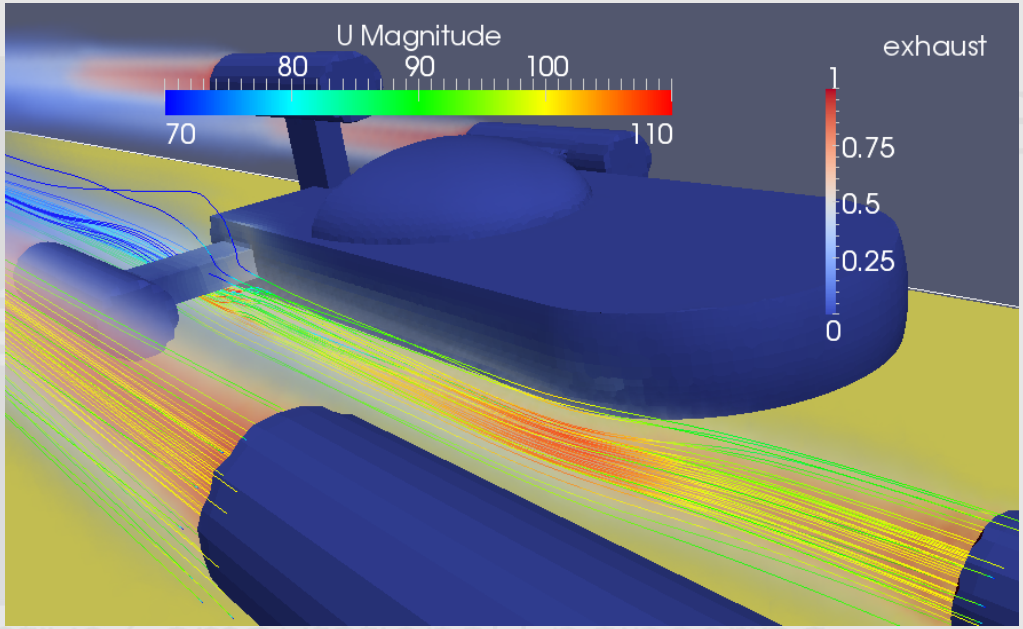


Figure : Distribution of exhaust gas on Biggs landspeeder

Biggs not intoxicated

- The results indicate that the exhaust gas did not have significant concentrations near the driver
  - Concentrations behind the speeders in the canyon is much higher
  - But to be sure we'd have to probe behind the windshield
- So Biggs probably wasn't impaired by his childhood adventures
  - He just had bad luck at the Death Star
    - Or the scriptwriter didn't like him
- But we see high fluctuations in the solution
  - So we have a look at the change of the velocity between timesteps

Calculating the change of the velocity

Stored variables

- Variables declared in `storedVariables` keep their value between timesteps
  - Can be used in `variables` before defined
- Need a `initialValue` for startup

controlDict

```
velocityChangeField {
  type expressionField;
  fieldName changeU;
  autowrite true;
  expression "changeOfU";
  variables (
              "changeOfU=U-oldU;"
              "oldU=U;"
              );
  storedVariables (
    {
        name oldU;
        initialValue "vector<brk>
            <cont>(80,0,0)";
    }
  );
}
```

Change of velocity

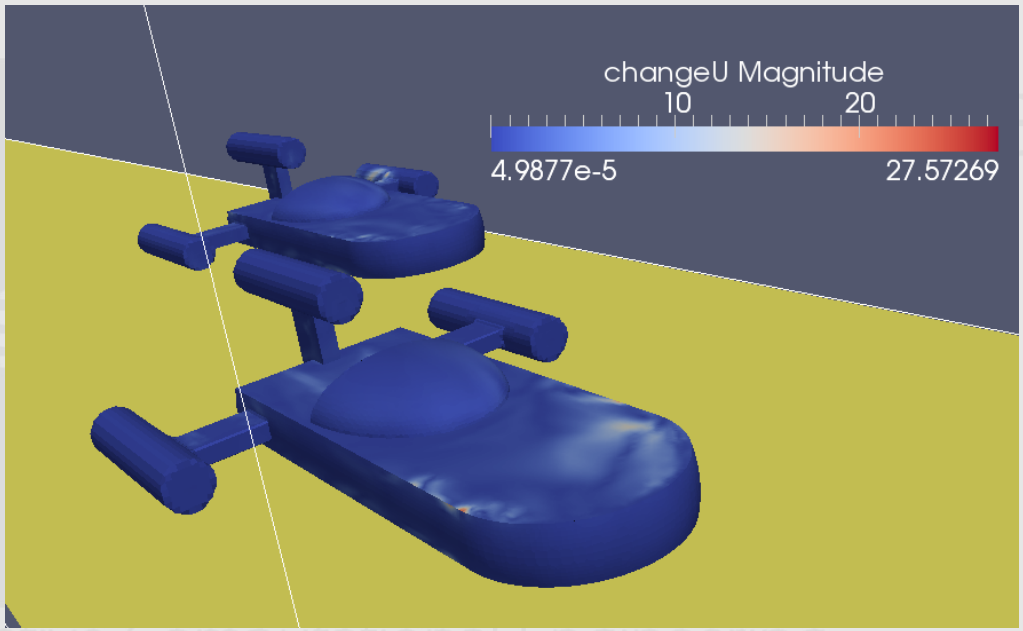


Figure : Change of velocity on the landspeeder surfaces

Change of velocity during simulation

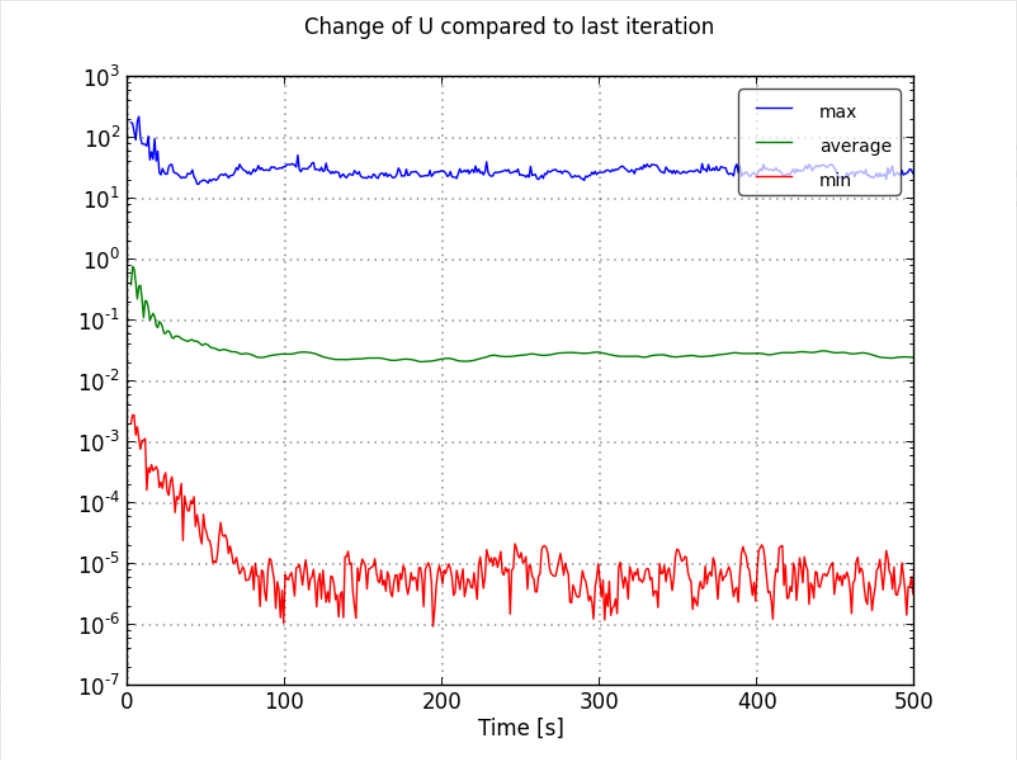


Figure : Graph of velocity change between timesteps

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

Position of biggest velocity change

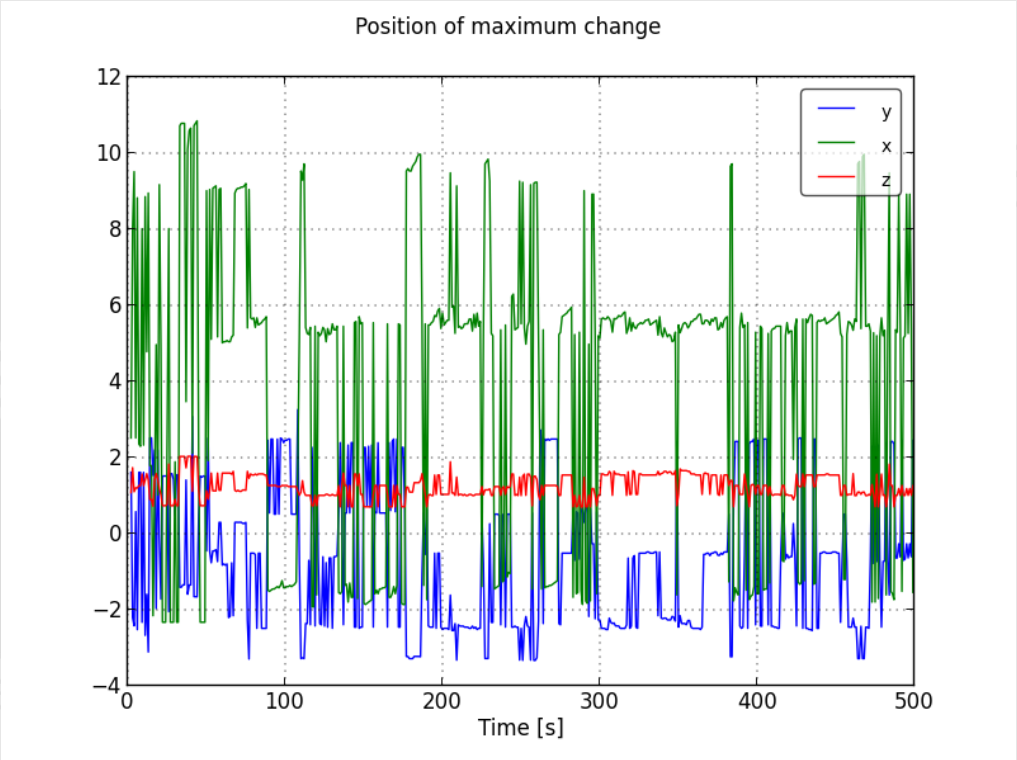


Figure : Position of the cell with the largest velocity change

Calculating distribution

- swak has a function object that allows the calculation of the distribution of a variable
- Three expressions that need to be defined

  expression The value of which the distribution is calculated

  weight Corresponding weight for each value (for instance the cell volume)

  mask Logical expression that says whether the value is used for the distribution
- Bin-size has to be specified
  - If this generates to many bins the size is rescaled
- Written data is
  - the distribution
  - the accumulation
  - A timeline of the main properties of the distribution (minimum, maximum etc)

Writing at specific intervals

- This works for all function objects based on `simpleFunctionObjects`
- New `outputControlMode` named `deltaT`
  - Desired output interval specified by `outputDeltaT`
- Function object is only evaluated if close enough to the "next" output time
  - Time is not manipulated so the intervall is **not** exactly fulfilled

Specifying two distributions

Volume weighted

```
velocityChangeDistribution {
    type swakExpressionDistribution<brk>
        <cont>;
    outputControlMode deltaT;
    outputDeltaT 10;
    valueType internalField;
    verbose true;
    expression "changeU";
    weight "vol()";
    mask "true";
    writeTimeline true;
    writeDistribution true;
    distributionBinWidth 0.1;
}
```

Cell weighted

```
velocityChangeDistributionCell
    <cont> {
    $velocityChangeDistribution<
        <cont>;
    weight "1";
}
```

Distribution of velocity change

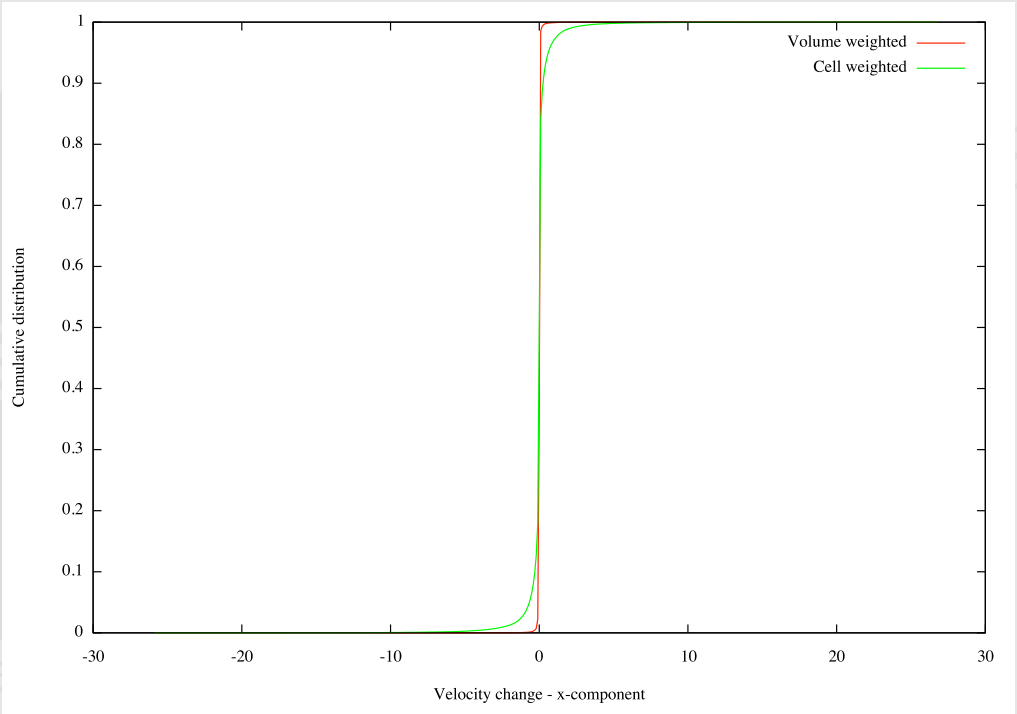


Figure : How common are large velocity changes?

Checking whether non-orthogonality is to blame

Functions

- `mqFaceNonOrth` is a plugin-function from the mesh quality plugin
- `lcFaceMaximum` is a plugin-function that calculates cell-values by looking at the maximum of a surface-field

controlDict

```
velChangeNonOrth {
    $velocityChangeDistribution;
    expression "lcFaceMaximum(<brk>
        <cont>mqFaceNonOrtho())";
    mask "mag(changeU)>5";
}
totalNonOrth {
    $velChangeNonOrth;
    mask "true";
}
```

Distribution of cell non-orthogonalities

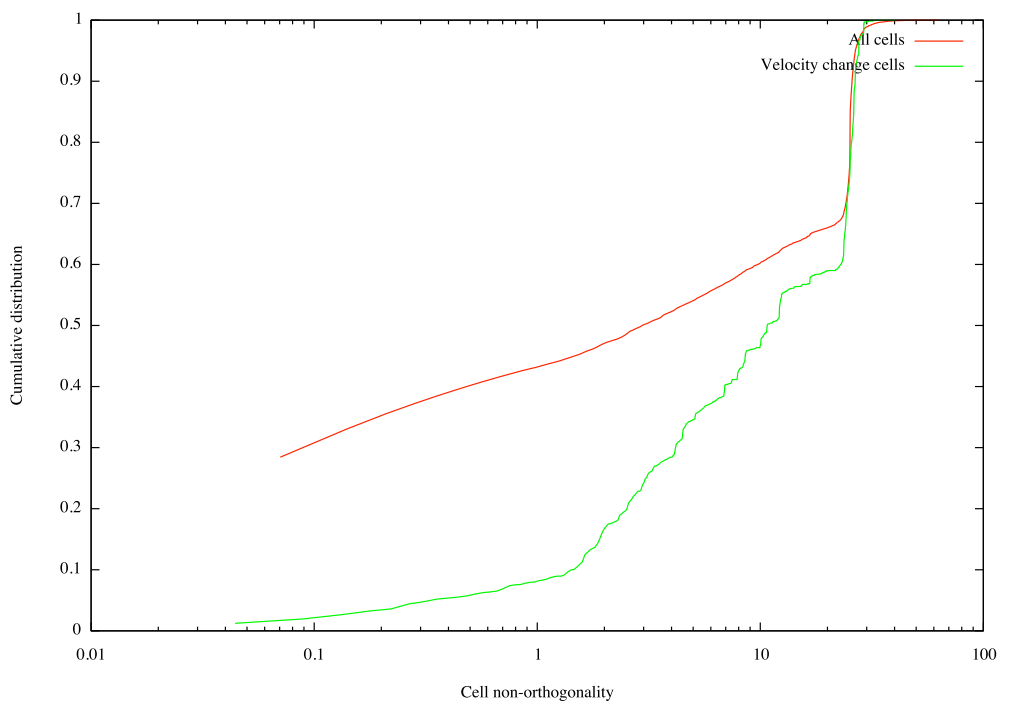


Figure : Velocity change correlates with non-orthogonality

Source of the fluctuations

- The distribution graph indicates that there is a **correlation** between non-orthogonal cells and velocity change
  - But **correlation** is not the same as **causality**
- There are also correlations with
  - cell size
  - skewness
- So we're not sure
  - But similar fluctuations can be seen in the `motorCylce`-tutorial

Strömungsforschung GmbH
ICE
Innovative Computational Engineering

What isn't shown

- The case also demonstrates dynamically generating function objects
  - A python-script generates text specifications for additional function objects
    - Sample planes in regular distances
    - Evaluation of the average pressure/exhaust concentration on these planes
  - Values are accumulated and written
- If we still got time I'll show you

Further information

- In the swak-distribution there is a file `Documentation/swak4FoamReference.org`
  - This document is work in progress
- The first part is already finished:
  - Expression syntax
  - **General** options for the parsers
- What is missing:
  - Description of the available function objects, utilities, boundary conditions
- Document will be updated in parallel to the sources

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

Outline