OpenFOAM

The open source CFD toolbox

Lessons in OpenFOAM – Tips & Tricks
10th OpenFOAM conference

OpenCFD Ltd

Notes to workshop

Contents

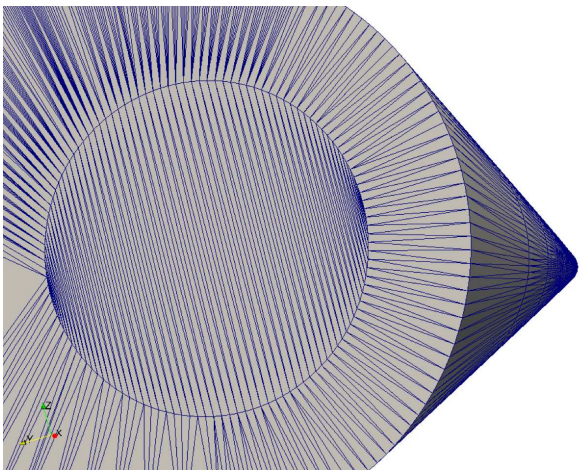
1 SnappyHexMesh

Upgrade overview

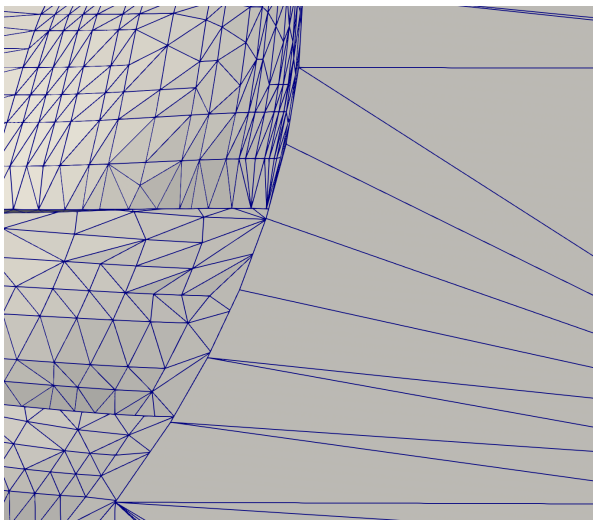
- Leaking mesh treatment development
- Layer additional improvements
- Performance improvements

Leak detection

- One of the important moments during meshing is to recognize which part of the mesh to keep and which to delete at the end of the castellation
- We claim `snappyHexMesh` can work with dirty meshes including holes in the triangulated surface description
- Such holes does not need to be missing triangles, much more often it is a topologically inconsistent mesh from the opposite side of the feature



topologically correct edge definition



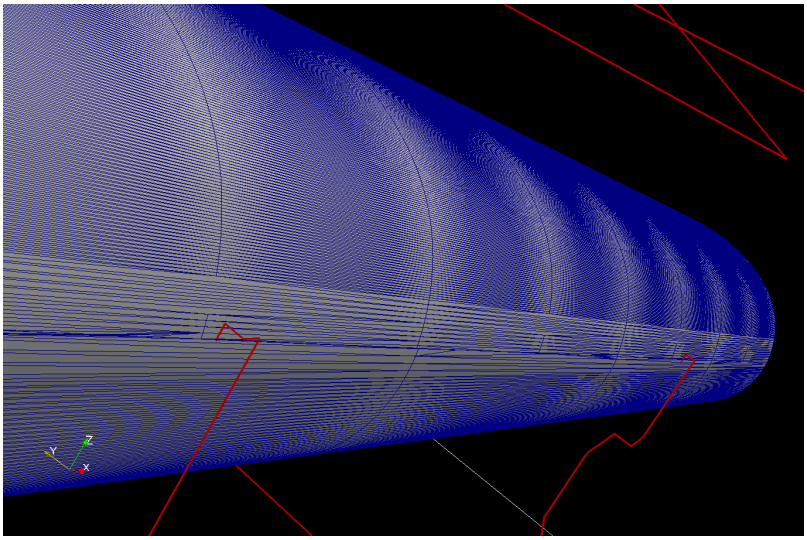
topologically incorrect edge definition

- The leak detection algorithm is turned on automatically once we specify `locationInsideMesh` and `locationOutsideMesh` keywords with their appropriate locations
- Walking algorithm searching for the closest distance between those two points will discover a leak in the triangulated surface and write down the path created from the cell centres to the VTK file for visual inspection in `ParaView`
- also Error message is issued

  frame frame

```
--> FOAM FATAL ERROR:
Location in mesh (2.899 1.9126 0.547) is inside same mesh region 0 as one of the locations
    outside mesh 2((0.5 0.3 0) (0.2 0.7 0))
    Dumped leak path to "/home/matej/PROJECTS/AIAA/OFrun/postProcessing/1/leakPath_leakPath.vtk"
```

- In `ParaView` the path can be checked to identify a place where surface must be repaired or redefined



Leaking mesh treatment

- In the previous versions of OpenFOAM we have already introduced leak-detection algorithm triggered on by specification of `locationOutsideMesh` which, should the leak be present, marked a path into a VTK file for visual inspection.
- Together with gap detection and refinement we have already powerful tools to work with.
- New boolean keyword `useLeakClosure` is turning on new leak-closure algorithm.

```
refinementSurfaces
{
    "sphere.*"
    {
    // Surface-wise min and max refinement level
        level (6 6);
    // Optional level at which to start early checking for leaks
        leakLevel 2;
    }
...
```

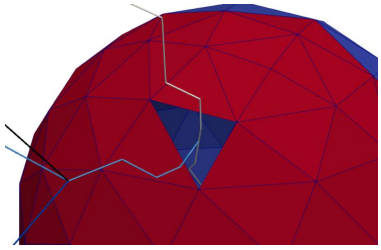
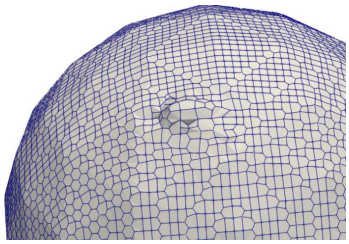
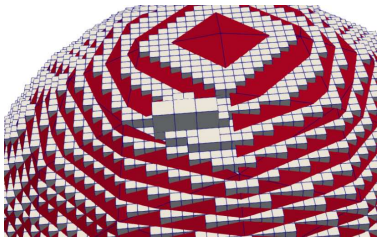
- When the optional `leakLevel` keyword is used, the detection starts from the specified level, restricting the search from the top, thus speeding the process. Without the specification the algorithm is starting from the top level.
- When the leak is smaller than the surface refinement level, no gap is identified.

Output

```
Removing mesh beyond surface intersections
------------------------------------------

Dumped leak path to ".../postProcessing/1/leakPath/leakPath.case"
--> FOAM Warning :
    From void Foam::meshRefinement::zonify(const bool, const Foam::label, const Foam::label, const Foam::pointField &, const Foam::w
    in file meshRefinement/meshRefinementBaffles.C at line 2990
    Locations in mesh 1((1.5 1.5 1.5)) connect to one of the locations outside mesh 1((3.8 3.8 3.8))
Setting cellZones according to locationsInMesh:
Location : (1.5 1.5 1.5)
    cellZone : none

For cellZone none found point (1.5 1.5 1.5) in global region 1 out of 2 regions.
--> FOAM Warning :
    From void Foam::meshRefinement::nearestFace(const Foam::labelUList &, const Foam::bitSet &, autoPtr<Foam::mapDistribute> &, Foam
    in file meshRefinement/meshRefinement.C at line 517
    Did not visit some faces, e.g. face 0 at (0.25 0.0625 0.0625)
Keeping all cells containing inside points
Selected for keeping : 43805 cells.
Detected 50 faces out of 17247 for which the default patch 1 will be used
```

```
Edge intersection testing:
    Number of edges             : 151962
    Number of edges to retest   : 17247
    Number of intersected edges : 19011
```

- Please help us to test this new feature under various conditions and report problems on the git platform webAddresshttp://develop.openfoam.com

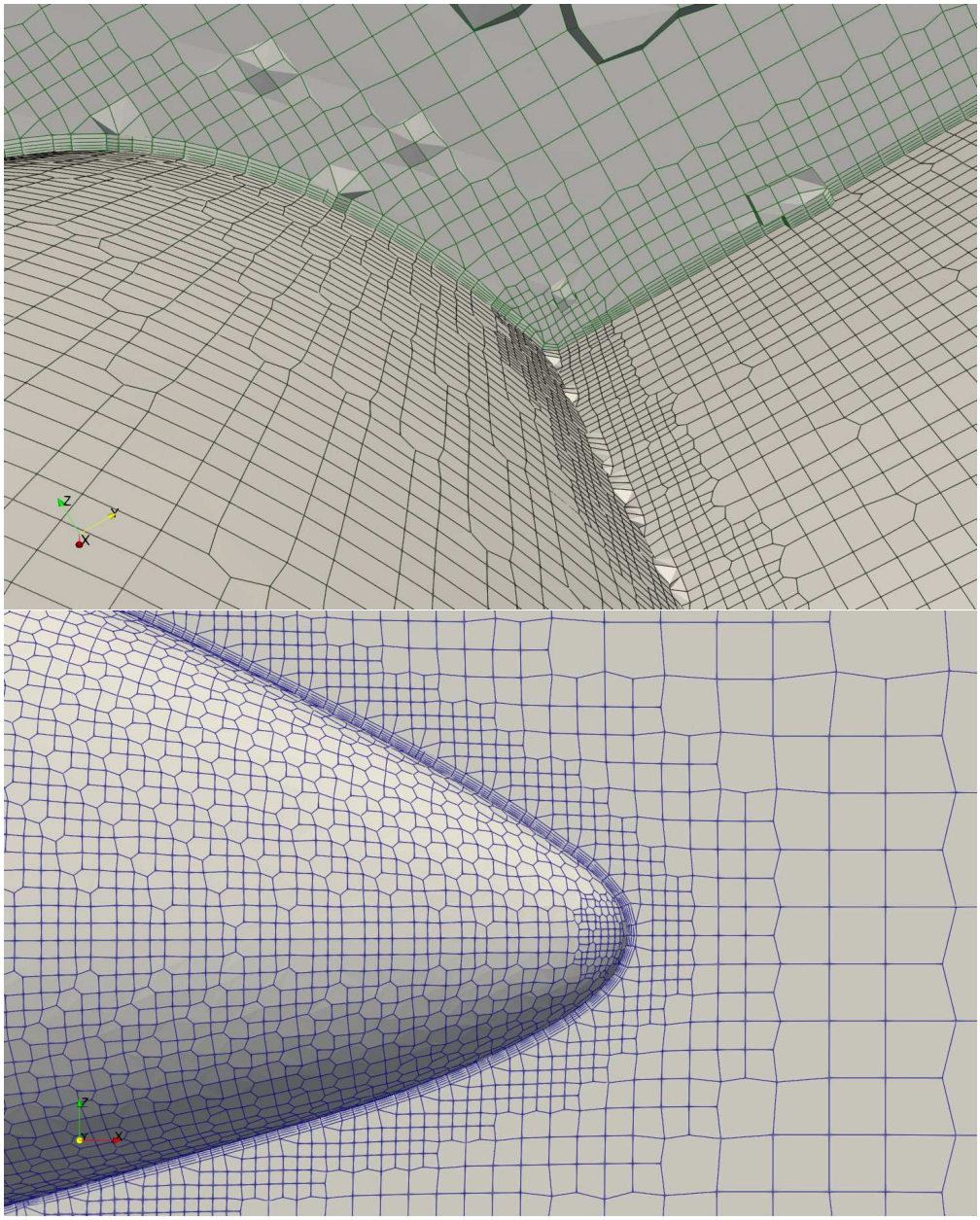




New thickness model from v2012

- Thickness to the layers can be specified either in relative size with respect to the local mesh size or in absolute thickness ignoring local refinement
- Using relative thickness seems more sensible option but quite often lead into a layer collapse on the boundary between different sizes
- New method was devised were the thickness can be specified in a hybrid (mixed) way
- New kyeword `thicknessModel` can be specified to switch one of the following options on:
  - firstAndOverall;
  - firstAndExpansion;
  - finalAndOverall;
  - finalAndExpansion;
  - overallAndExpansion;
  - firstAndRelativeFinal.
- The last option setting first layer in absolute measure and the last in relative sizing allows to target for specific yPlus value while leaving freedom to react on the local refinement. This seems to avoid the collapse on the edge of local refinement.

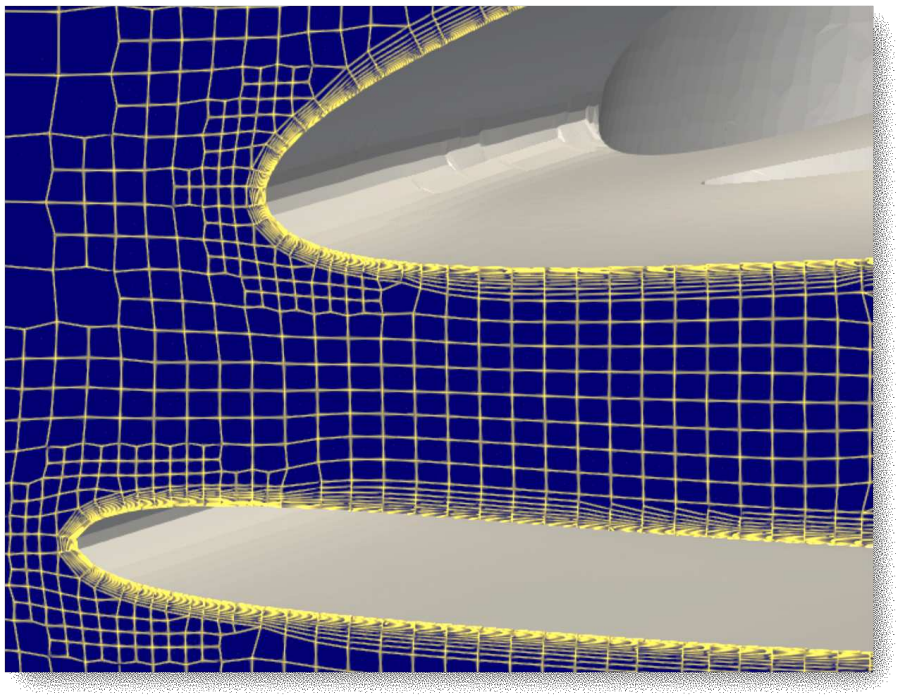
Example setup:

```
addLayersControls
{
    ...
    thicknessModel  firstAndRelativeFinal;
    firstLayerThickness  0.1e-3; //in [m]
    finalLayerThickness  0.2; // relative size
    ...
```



Layer addition improvement

- New parameter nOuterIter sets the number of passes of the snappyHexMesh through the addLayer part.
- In other words there will be n number of times passes to build gradually layers.
- It is a new feature and needs to be tested.



Performance

- Performance improvements specifically in the castellation part of the process is a reason to update to the latest version v2206
- According to our meshing tests (on DriveAer car model for mesh for external aerodynamics) the speed-up compared to the v2106 the castellation is faster for approx. 38 per cent and the overall performance is improved by 18 per cent in execution time
- Of course when you learn about our improved layer coverage, you might consider building more layers spending on meshing more time in total.
- Time spent on mesh quality improvement is an investment into quality of the results.

2 Expression language

Introduction

- Expression language has been introduced to improve flexibility of the setup
- Expressions can be used to
  - provide input to various setup dictionaries in `system` and `constant` directories,
  - describe boundary conditions,
  - provide more flexible initial conditions
- Expressions are strings following predefined grammar parsed into operations for evaluation.
- Unlike the `calc` directive of a dynamic coding, that is compiled into a C++ library, the expressions are evaluated by parser almost instantly.

- Parser operation comprises 3 stages:
  - scanner splits the string into individual input tokens (function names, operators ...)
  - grammar parser handles relationships between the components
  - driver acts as an intermediate for the scanner and grammar parser when retrieving or storing OpenFOAM fields, or obtaining mesh-relevant quantities such as cell centers etc.
- The expression language could be used not only for simple algebra, but also to manipulate with OpenFOAM fields

frame frame

```
radius  0.5; //m
angle   45; //deg
yPos    #eval "\$radius * cos(degToRad($angle))";

Ux      10;
turbInt 0.12;
Uinlet  ($Ux 0 0);
tke_in  #eval{ 1.5*sqr($Ux*$turbInt) };

flx     #eval "alpha.air * mag(rho * U)";
```

2.1 Grammar overview

Data types

- Expressions support many of the C++ syntax (including comments) as well as OpenFOAM specific functions and types.

bool	boolean field result of logical operation
scalar	floating point values, scalar fields
vector	vector fields in Cartesian global CC
tensor	tensor field with 9 elements
symmTensor	a six-component symmetrical tensor
sphericalTensor	a single component spherical tensor

Operators

$+ - */$	arithmetic operations
&	Inner (dot) product for vectors and tensors (result. scalarField)
^	Cross product of two vectors (result. vectorField)
%	Modulo operator
&& , ||	Logical and and or operators
−	Unary negation
!	Logical negation
<>>=<=	Relational comparisons
==! =	Equality and inequality-operators
? and :	Ternary operator for cond ? a : b.

Identifiers

- Expression grammar is using the same requirements as OpenFOAM code for the variable names. Alphanumerics with underscores are permitted as well as a dot (if not at the start of the name)
- Other general punctuation can be used as a part of the name if quoted.

- There is no semantic difference between single and double quotes.

  frame frame

```
// Good quoting!
pos("alpha.x") * 0.4*neg('alpha.xx')
"field:a-b"  //- name of the single field
"sin(angle_top)"  //- name of the field
```

- Two last examples are not parsed as functions or operations but parsed as variable names.

Composing data types

- In expressions we do not need to declare the types, we need to use correct composition syntax instead

  frame frame

```
//- composing e.g. position vector
v1 = vector(1, magSqr(pos().y()), cos(degToRad($angle)));

//- composing zero vector
v0 = vector(Zero)

//- composing unit tensor
t1 = tensor::I

//- composing symmetry tensor
t2 = symmTensor($xx, $xy, $xz, $yy, $yz, $zz)
```

Boolean tricks

- Keyword `bool` can be used to force a boolean conversion of scalar values. The threshold of $-/+$ 0.5 isd used to define `true/false`
- Assuming scalar values of 0 and 1 are the truth we can use the bool to various rounding and interpolation

  frame frame

```
bool(-11)     ==> true
bool(-0.25)    ==> false
bool(0.38)     ==> true

//- expression
mask = bool(pos(x))
//- corresponds to:
mask = (pos(x) > 0.5)
```

Mesh information

- Functions used without arguments

Function	Domain	Description
`pos()`	P,V	Native positions (P: face centres, V: cell centres)
`pts()`	P,V	Point positions (P: face points, V: mesh points)
`fpos()`	V	Face centres
`area()`	P,V	Face area magnitude
`face()`	P,V	Face areaNormal vectors
`vol()`	V	Cell volumes

Accessing Zones

- Set of functions can be used to access the zones predefined in the volumetric mesh.
- Functions are returning true/false for the given zone
- These are: `cset(NAME), fset(NAME), pset(NAME), czone(NAME), fzone(NAME), pzone(NAME)`
- Functions `weightAverage(), weightSum()` work as expected; if point field is given, they behave as simple average and sum

Working with fields

- Fields registered in the memory in objectRegistry are found via lookup function.
- We have to be careful when using the functions. While frame frame

  ```
  pos(U.x())
  ```

- is returning a field of ones and zeros depending on the x–coordinate value, taking the `pos` function as a unary one
- On the other hand, the example frame frame

  ```
  pos()
  ```

- will return the position field of the cell centres.

Variables

- Variables are specified in the optional dictionary `variables` in the following form:

  frame frame

  ```
  variables
  (
      "varName1 = expression1"
      "varName2 = expression2"
      ...
  );
  ```

- When specifiing the variable, we can add the specification of the patch or a volume as in `varName{patchName}`
- Let us see some examples

  frame frame

  ```
  variables
  (
      "pInlet{inletPatch} = weightedAverage(p)"  // getting aver. value
      "pOutlet2{outlet2patch} = p"  // getting the fixedValue
      "temp2aver{outlet2patch} = weightedSum(rho * T) / weightedSum(rho)"
  );
  ```


Example use of expressions

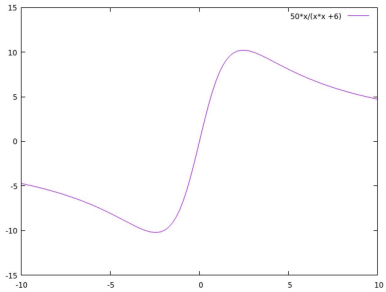
- We can use the expression language in three accasions:
  - `#eval` statements in various dictionaries
  - `exprFixedValue` – fixed value BC
  - `exprMixed` – mixed type BC
  - `setExprFields` – to set initial volumetric field
  - `setExprBoundaryFields` – to set initial `fixedValue` BC fields
- Let us see some examples

Example use: eval statement

```
//- initial condition
Umag         10;
Uin          #eval{ vector(Umag, 0, 0) };
turbInt      0.1;
D_in         0.0254;
L            #eval{ $D_in * 0.07 };

k_in         #eval{ 1.5*sqr($Umag*$turbInt) };
eps_in       #eval{ pow(0.09,0.75)*pow($k_in,1.5)/$L };
omega_in     #eval{ pow(0.09,-0.25)*sqrt($k_in)/$L };
```

Example use: `uniformFixedValue` BC

```
    inlet
    {
        type            uniformFixedValue;
        uniformValue
        {
            type            expression;

            // optional variables for use within the expression
            variables
            (
                "scale = 50"
            );

            // A step function
            expression
            #{
            // arg() returns current time
            -1*face()/mag(face())*scale*arg()/(sqr(arg()) + 6.0)
            #};
        }
    }
```



- You can test it on simpleFoam (change `writeInterval`) to 5

`exprFixedValue`

- Boundary condition designed directly to the purpose of using expression language could be replacing `codedFixedValue` in many cases
- One of the advantages is *e.g.* easy reference to the various patches or volumes in the domain

```
    outlet1
    {
        type            exprFixedValue;
        value           $internalField;

        valueExpr       "0.5*(pInlet + pOutlet2)";
        variables
        (
            "pInlet{inlet} = weightAverage(p)"
            "pOutlet2{outlet2} = p"
        );
    }
```


`exprFixedValue`

- Another example shows a parabolic profile for the `pitzDaily` tutorial case

```
    inlet
    {
        type            exprFixedValue;
        value           uniform (10 0 0);

        valueExpr  "nf*(-6.0*Umean*length/h*(1.0- length/h))";
        variables
        (
           "h = 0.0254"
           "Umean = 10"
           "length = pos().y()"
           "nf = face()/mag(face())"
        );
    }
```

Example use: `setExprFields`

```
expressions
(
    T
    {
        field       T;
        dimensions  [0 0 0 1 0 0 0];

        constants
        {  centre (0.21 0 0.01); }

        variables
        ( "radius = 0.1" );

        condition
        #{
            // Within the radius
            (mag(pos() - $[(vector)constants.centre]) < radius)
            // but only +ve y!
          && bool(pos((pos() - $[(vector)constants.centre]).y()))
        #};

        expression
        #{
            300 + 200 * (1 - mag(pos() - $[(vector)constants.centre]) / radius)
        #};
    }
);
```

3 Dynamic Mesh

Dynamic mesh usefull combination

- Can we combine solid body motion with mesh refinement?
- Yes we can via using of `solver` dictionary entry in `<constant>/dynamicMeshDict`
- In this dictionary we may specify a `motionSolver` to the whole domain. This is where we specify the solid body motion.
- In the main "body" of the setup we then may specify the refinement of the mesh.

```
solvers
{
VF
    {
        motionSolverLibs        (fvMotionSolvers);

        motionSolver            solidBody;

        solidBodyMotionFunction oscillatingLinearMotion;
        amplitude               (0.1 0 0);
        omega                   #eval{ 2*pi() };
    }
}
```

- The movement is applied the the whole body of the domain
- User may pick any of the predifined solid body motions:
  - linearMotion,
  - rotationalMotion,
  - oscillatingLinear motion,
  - oscilatingRotating motion,
  - axisRotatiion motion,
  - multiMotion,
  - SDA,
  - tabulated6DoFMotion,
  - drivenLinearMotion.

Refinement specification

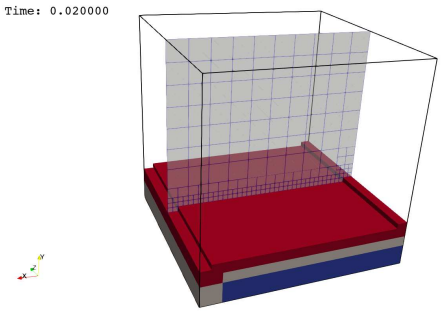
- Automatic refinement of the mesh is then specified below:

```
// How often to refine
refineInterval  1;

// Field to be refinement on
field           alpha.water;

// Refine field inbetween lower..upper
lowerRefineLevel 0.001;
upperRefineLevel 0.999;

// If value < unrefineLevel unrefine
unrefineLevel   10;

// Have slower than 2:1 refinement
nBufferLayers   1;

// Refine cells only up to maxRefinement levels
maxRefinement   2;

// Stop refinement if maxCells reached
maxCells        200000;
```


- The setup sets the mesh to be treated by the dynamic refinement library while the motion library used is the `solidBody` one.
- Output from the solver is specifying used models:

```
Selecting dynamicFvMesh dynamicRefineFvMesh
Selecting motion solver: solidBody
Applying motion to entire mesh
Selecting solid-body motion function oscillatingLinearMotion
```

- Solver `interFoam` is then executing both dynamic mesh capabilities
- Whole domain is moving while the mesh around the interface is refining (and unrefining)



Refinement settings

- Refinement can be driven by scalar concentration, for multiphase flows it is typically the `alpha.<fluid>` scalar
- In other cases quite popular request is to follow a gradient of the field. In such case user should use the function object `grad` and then pass the obtained vector field into another function object `mag` to produce a scalar field we can use in a dynamic mesh as an indicator for mesh refinement.
- Using a `coded` function object we can also create any indicator for the refinement we would need.

4 Upgrade info

histogram function object

- Very useful tool during the case debugging is the minMax function object.
- However there is only limited information transferred.
- New function object was introduced to improve the situation - histogram
- This function object gives you more detailed information and with the help of *e.g.* gnuplot we may plot real histogram.

Latest upgrades user should know about - Coordinate systems

- Dictionary entries for specification of the local coordinate system (*e.g.* in porous zone setup) are easier to specify and understand.
- The older long names axesRotation, EulerRotation and STARCDRotation are now reported as aliases for the corresponding short names axes, euler and starcd, respectively.
- The transform or coordinateSystem entries now accept an inline specification of the rotation type

```
transform
{
    origin  (0 0 0);
    rotation axisAngle;
    axis    (0 0 1);
    angle   45;
}
```

```
transform
{
    origin  (0 0 0);
    rotation
    {
        type  axisAngle;
        axis  (0 0 1);
        angle 45;
    }
}
```

- Keyword `none` can be used for pure translation or use the implicit axes specification:

```
transform
{
    origin   (1 2 3);
    rotation none;
}
```

```
transform
{
    origin  (1 2 3);
    e1      (1 0 0);
    e3      (0 0 1);
}
```

Co-ordinate system rotation methods

- Transformation types available in `coordinateRotation` class (see Doxygen for details) :
  - `axesRotation`
  - `axisAngleRotation`
  - `identityRotation`
  - `STARCDrotation`
  - `cylindrical`
  - `EulerCoordinateRotation`
  - `EulerCoordinateRotation`
- `axesRotation` is given by:

  - Defined by a combination of vectors (e1/e2), (e2/e3) or (e3/e1)
  - Any non-orthogonality will be absorbed into the second vector

```
axesRotation
    {
        type        axes;
        e1          (1 0 0);
        e2          (0 1 0);
    }
```

- `cylindrical` is given by:

```
coordinateRotation
        {
            type        cylindrical;
            e3          (0 0 1);
        }
```

- `EulerCoordinateRotation`

  - The 3 rotations are defined in the Euler convention (around Z, around X' and around Z')
  - Rotation angles are in degrees, unless otherwise explicitly specified
  - More details on: `http://mathworld.wolfram.com/EulerAngles.html`

```
coordinateRotation
    {
        type    euler;
        angles  (0 0 180); // previously in rad
    }
```

- `axisAngleRotation`

```
rotation
    {
        type        axisAngle;
        axis        (1 0 0);
        angle       90;
        degrees     yes;
    }
```

Latest upgrades user should know about - Sampling

- Most of the sampling planes are created with a `pointAndNormal` method, therefore the `planeType` has now default value of `pointAndNormal`. The whole specification can be then simplified:

```
slice
{
    type        cuttingPlane;
    planeType   pointAndNormal;
    pointAndNormalDict
    {
        point   (0 0 0);
        normal  (0 0 1);
    }
}
```

```
slice
{
    type    cuttingPlane;
    point   (0 0 0);
    normal  (0 0 1);
}
```

Developer upgrade guide – thermotools

- New `thermoTools` library was introduced as a part of restructuring of the compressible solvers
- At the moment it includes various derived boundary conditions, but will be extended in the future.
- New solvers using existing `compressibleTurbulenceModels` should contain a link to the library.

- Currently only `EXE_LIBS` entry must contain its specification in `MAKEoptions` file:

```
EXE_LIBS = ... \
    ...
    -lcompressibleTurbulenceModels \
    -lthermoTools \      # <<---  NEW ENTRY
    -lcompressibleTransportModels \
    ...
```

Developer upgrade guide – Field normalise

- `Field` container has now new method `normalise()`
- For `vector` and `solveVector` it corresponds to `vector::normlise`, but applied to each element.

```
// NEW METHOD
fld.normalise();

// OLD METHOD
 fld /= mag(fld) + VSMALL;
```


5 Solver performance

Performance tricks – Caching gradients

- In `fvSolution` we can set the option to cache some intermediate fields
- Typically we choose to cache the gradients

```
cache
{
    grad(U);
}
```

- The question is what effect it has on a performance and whether caching other gradients could help as well. Typical choice could be a pressure gradient, which is part of the N-S equations.
- Test performed on DrivAer model for external flow around car (our standard test with 80 mil. cells) shows speedup of 39 per cent with velocity gradients. The speedup was consistent on parallel runs of different sizes.
- Caching gradient of pressure, however, does not bring any effect.

GAMG Solver Settings for Transient Cases

- When using `PISO` velocity—pressure coupling algorithm as well as with `PIMPLE` we need to set the `pFinal` – setup for pressure final loop. Normally we set here `relTol` to 0. This setup helps to maintain high level of convergence at the end of the time step.
- With the similar spirit we could also set specific settings to the `coarsestLevelCorr` sub–dictionary which sets the solver parameters for the coarsest level.
- GAMG solver is using a V-cycle, which is computationally cheap compared to the correction on the coarsest level in GAMG.
- It seems to make sense to make a rough solution on the coarsest level which will cost us more V-cycles.
- Example of such setup can be found in `$FOAM_TUTORIALS/incompressible/pisoFoam/LES/motorBike`

```
pFinal

    p
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.05;

        smoother        GaussSeidel;


        nCellsInCoarsestLevel 50;
    }
```

```
pFinal
    {
        $p;
        relTol          0;

        // Explicit specify solver for coarse-level
    correction to override
        // solution tolerance
        coarsestLevelCorr
        {
            // For limited residual reduction (relTol
    ) PPCR behaves better
            // than PCG (or PPCG)
            solver          PPCR;   //PCG;
            preconditioner  DIC;
            relTol          0.05;
        }
    }
```

- `PPCR` solver is the Preconditioned pipelined conjugate residuals solver for symmetric lduMatrices using a run-time selectable preconditioner.
- Reference: P. Ghysels, W. Vanroose. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm., Paul Eller, William Gropp Scalable Non- -blocking Preconditioned Conjugate Gradient Methods

GAMG Solver Settings for Steady state Cases

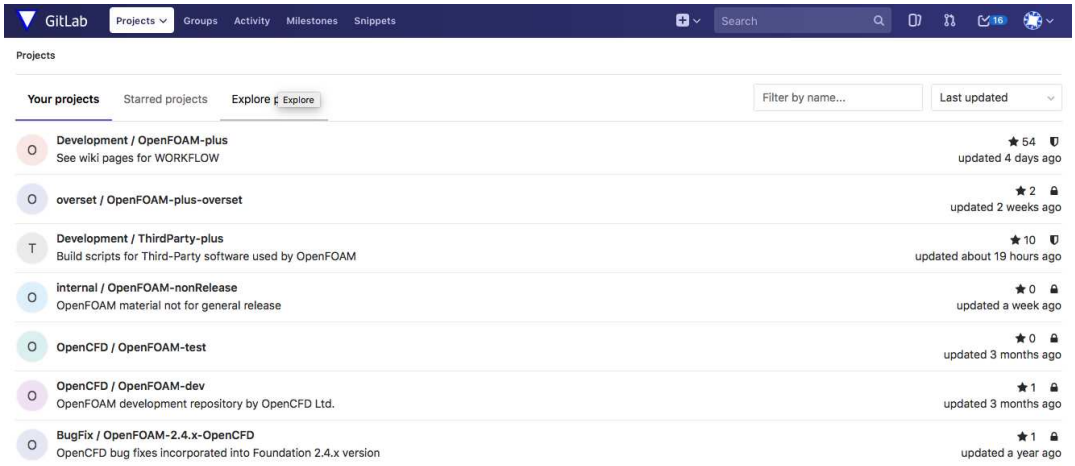
- For steady state simulations we can also set the different solver parameters for the `coarsestLevelCorr`
- The default `PCG` with `DIC` preconditioner can be overwritten with `PPCG` or `PPCR`

```
p
    {
        solver          GAMG;
        relTol          0;
        tolerance       1e-6;
        ...

        coarsestLevelCorr
        {
            // For limited residual reduction (relTol) PPCR behaves better
            // than PCG (or PPCG)
            solver          PPCR;   //PCG;
            preconditioner  FDIC;
            relTol          0.1;
            maxIter         30;
        }
    }
```


6 GitLab - bug reporting

Using a GitLab

- OpenCFD is using a GitLab – web based git repository including also issue tracking and wiki
- GitLab is located at: `develop.openfoam.com`
- To access the service, free registration is necessary (confirmation email will be sent to the registered email address)



- There are several projects you can view
- Bug-fixing branch of current release (OpenFOAM-plus and ThirdParty-plus)
- Community branch – offers a git space for selected projects which needs to be compiled separately to be used
- Current list comprise:
  - catalyst
  - cfmesh
  - isoAdvector — integrated
  - ihccantabria — integrated
  - adiosWrite — Adaptable IO System — integration of the ADIOS libraries speed up parallel IO operations
  - and few others . . .

- User may browse or search in the list of issues and the following discussion
- New issue could be reported (the more and more specific information you will provide the better)
- GitLab may be as well used to pull the latest public version of the code using a git command
- To be able to access the git repository, the ssh key on a given computer must be registered in the profile settings under the ssh keys
- Then the user can use command `git clone <copiedURL>` in the terminal