Rotating machinery training at OFW16

Håkan Nilsson

Mechanics and Maritime Sciences/Fluid Dynamics,
Chalmers University of Technology,
Gothenburg, Sweden
https://www.chalmers.se/sv/personal/Sidor/hakan-nilsson.aspx
http://www.tfd.chalmers.se/%7Ehani
https://www.abcfd.se

Contributions from:
Martin Beaudoin (IREQ, Hydro Quebec)
Maryse Page, IREQ, Hydro Quebec
Hrvoje Jasak, Wikki Ltd.

2021-06-09

Code

- I have used a TurboWG version of FOAM-extend-4.1, nextRelease branch (a.k.a. 5.0), available at `https://sourceforge.net/projects/turbowg/`.
- Developments aim to be added to FOAM-extend.
- Download and compile:

```
git clone -b TurboWG_experimental \
    https://git.code.sf.net/p/turbowg/foam-extend-5.0 foam-extend-5.0_TurboWG_experimental
cd foam-extend-5.0_TurboWG_experimental
sed -i s/"5\.0"/"5\.0_TurboWG_experimental"/g etc/bashrc
echo "export PARAVIEW_SYSTEM=1" >> etc/prefs.sh #Avoid having to type "Y" at compilation
. etc/bashrc
./Allwmake.firstInstall >& log
./Allwmake.firstInstall >& log
```

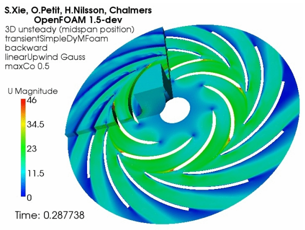
- Revision used here: ccb3576.

What's this training about?

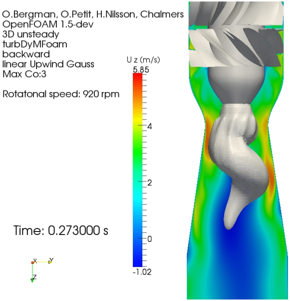
- The focus is on *rotating machinery* and functionality that is related to rotation
- We will investigate the theory and application of SRF, MRF, DyM, coupling interfaces, and other useful features
- We will use the `axialTurbine` tutorials to learn how to set up and run cases

Note: There are also full cases in the Sig Turbomachinery Wiki

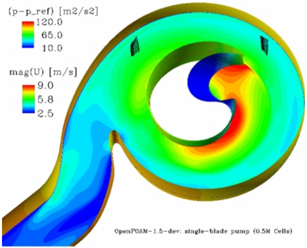
http://openfoamwiki.net/index.php/Sig_Turbomachinery



ERCOFTAC
Centrifugal Pump (ECP)



Timisoara
Swirl Generator (TSG)



Single Channel Pump
(SCP)

(these cases are being moved to `https://sourceforge.net/projects/turbowg/`)

Prerequisites

You know how to ...

- use Linux commands
- run the basic OpenFOAM tutorials
- use the OpenFOAM environment

Learning outcomes

You will know ...

- the underlying theory of SRF, MRF and DyM
- how to set up cases for rotating machinery

Fundamental features for CFD in rotating machinery

Necessary:

- Utilities for special mesh/case preparation
- Solvers that include the effect of rotation of (part(s) of) the domain
- Libraries for mesh rotation, or source terms for the rotation
- Coupling of rotating and steady parts of the mesh

Useful:

- Specialized boundary conditions for rotation and axi-symmetry
- A cylindrical coordinate system class
- Tailored data extraction and post-processing

Training organization

The rotation approaches (SRF, MRF, DyM) are presented as:

- Theory
- Tutorials - how they are set up
- Dictionaries and utilities
- Special boundary conditions

Single rotating frame of reference (SRF), theory

- Compute in the rotating frame of reference, with velocity and fluxes relative to the rotating reference frame, using Cartesian components.
- Coriolis and centrifugal source terms in the momentum equations (laminar version):

$$\nabla \cdot (\vec{u}_R \otimes \vec{u}_R) + \underbrace{2\vec{\Omega} \times \vec{u}_R}_{Coriolis} + \underbrace{\vec{\Omega} \times (\vec{\Omega} \times \vec{r})}_{centrifugal} = -\nabla(p/\rho) + \nu \nabla \cdot \nabla(\vec{u}_R)$$

$$\nabla \cdot \vec{u}_R = 0$$

where $\vec{u}_R = \vec{u}_I - \vec{\Omega} \times \vec{r}$
Derivation at: `http://openfoamwiki.net/index.php/See_the_MRF_development`

- I.e., we need a rotational speed and compute *relative* velocity.

The simpleSRFFoam axialTurbine tutorial

We first run the tutorial, and then investigate the details of it...

- Run tutorial:

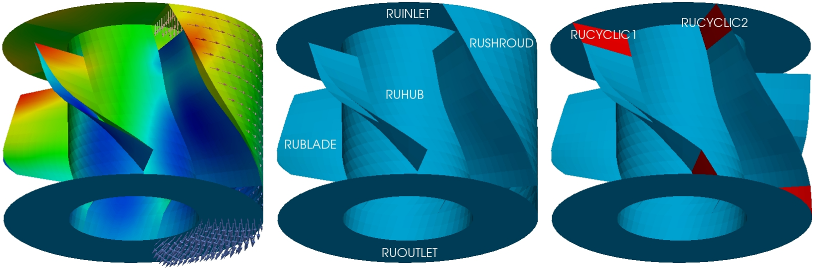
```
cp -r $FOAM_TUTORIALS/incompressible/simpleSRFFoam/axialTurbine_SRF $FOAM_RUN
cd $FOAM_RUN/axialTurbine_SRF
./Allrun >& log_Allrun &
```

- Look at the results:

```
paraFoam -builtin
```

Use `Angular Periodic Filter` to copy to all blade passages (mark `Block Indices`, set `Rotation Angle` to 72, set `Axis` to Z)

simpleSRFFoam axialTurbine tutorial results and boundary names



- `RUINLET` has an axial relative inlet velocity (`Urel`)
- `RUCYCLIC1` and `RUCYCLIC2` are cyclic, using `cyclicGgi`
- `RUBLADE` and `RUHUB` have zero relative velocity (`Urel`)
- `RUSHROUD` has a zero absolute velocity (`Urel` counter-rotating)
- `RUOUTLET` has an `inletOutlet` condition

The Allrun script (main features)

- Mesh generation
- Create GGI zones (necessary for parallel simulations)
- Run `simpleSRFFoam` solver (using solver settings)
- Post-process (e.g. calculate cylindrical velocity components)

Mesh generation and GGI zones

- The mesh is done with `m4` and `blockMesh`
  No need to bother about details, but for those interested:
  - Cylindrical coordinates are utilized (modified angle: 1/20)
  - The modified angle is transformed back to radians:
    `transformPoints -scale "(1 20 1)"`
  - The coordinates are transformed to Cartesian:
    `transformPoints -cylToCart "((0 0 0) (0 0 1) (1 0 0))"`
- GGI zones are created (see `setBatchGgi`):

```
setSet -batch setBatchGgi
setsToZones -noFlipMap
```

- In `system/decomposeParDict`:

```
globalFaceZones ( RUCYCLIC1Zone RUCYCLIC2Zone );
```

- The face zones are available for ParaView in the `VTK` directory

Solver settings: SRFProperties

The rotation is specified in `constant/SRFProperties`:

```
SRFModel  rpm;

axis (0 0 1);

rpmCoeffs
{
    rpm  -95.49; //-10 rad/s
}
```

Solver settings: Boundary condition for Urel

```
RUINLET
{
    type            SRFVelocity;
    inletValue      uniform (0 0 -1);
    relative        no;  // no means that inletValue is applied as is
                         // (Urel = inletValue)
                         // yes means that rotation is subtracted from inletValue
                         // (Urel = inletValue - omega X r)
                         // and makes sure that conversion to Uabs
                         // is done correctly
    value           uniform (0 0 0); // Just for paraFoam
}
RUSHROUD
{
    type            SRFVelocity;
    inletValue      uniform (0 0 0);
    relative        yes; // Counter-rotating
    value           uniform (0 0 0); // Just for paraFoam
}
```

Solver settings: cyclicGgi (boundary file)

- For patches `RUCYCLIC1` and `RUCYCLIC2`:

```
RUCYCLIC1
{
    type             cyclicGgi;
    nFaces           40;
    startFace        4020;
    shadowPatch      RUCYCLIC2;
    zone             RUCYCLIC1Zone;
    bridgeOverlap    false;
    rotationAxis     (0 0 1);
    rotationAngle    72;
    separationOffset (0 0 0);
}
```

```
RUCYCLIC2
{
    type             cyclicGgi;
    nFaces           40;
    startFace        4060;
    shadowPatch      RUCYCLIC1;
    zone             RUCYCLIC2Zone;
    bridgeOverlap    false;
    rotationAxis     (0 0 1);
    rotationAngle    -72;
    separationOffset (0 0 0);
}
```

- `rotationAxis` defines the rotation axis of the `rotationAngle`
- `rotationAngle` specifies how many degrees the patch should be rotated about its rotation axis to match the `shadowPatch`
- `separationOffset` is used for translationally cyclic patches

Solver settings: cyclicGgi (fields file)

- The field files (`epsilon`, `k`, `nut`, `p`, `Uabs`, `Urel`):

```
RUCYCLIC1
{
    type            cyclicGgi;
}
RUCYCLIC2
{
    type            cyclicGgi;
}
```

- Check all GGI setup ...

Solver settings: Check GGI setup

Activate `DebugSwitch`:

- In `Allrun`:

  `runApplication $application -DebugSwitches GGIInterpolation=1`

- In log-file:

```
Evaluation of GGI weighting factors:

From function GGIInterpolation::findNonOverlappingFaces ... GGIInterpolationWeights.C at line 834
   : Found 0 non-overlapping faces for this GGI patch

From function GGIInterpolation::findNonOverlappingFaces ... GGIInterpolationWeights.C at line 834
   : Found 0 non-overlapping faces for this GGI patch

From function GGIInterpolation::calcPartiallyCoveredFaces ... GGIInterpolationWeights.C at line 914
   : Found 0 partially overlapping faces for master GGI patch

From function GGIInterpolation::calcPartiallyCoveredFaces ... GGIInterpolationWeights.C at line 938
   : Found 0 partially overlapping faces for slave GGI patch
Largest slave weighting factor correction : 2.22045e-16 average: 5.82867e-17
Largest master weighting factor correction: 4.44089e-15 average: 1.12133e-15
```

Post-processing: ggiCheck

- The flux balance at the cyclic GGI pair is checked by activating the `ggiCheck` functionobject in `system/controlDict`:

```
functions
(
    // Compute the flux value on each side of a GGI interface
    ggiCheck
    {
        // Type of functionObject
        type ggiCheck;

        phi phi;

        // Where to load it from (if not already in solver)
        functionObjectLibs ("libcheckFunctionObjects.so");
    }
);
```

- Output in log file:

```
Checking flux phi GGI balance.
Cyclic GGI pair (RUCYCLIC1, RUCYCLIC2)
Area: 0.00221581 0.00221581 Diff = -1.30104e-18 or 5.87165e-14 %
Flux: 0.000219652 0.000219652 Diff = 2.94804e-13 or 1.34214e-07 %
```

Post-processing: Additional fields

- The solver automatically calculates and writes out the absolute velocity `Uabs`.
- We ask `foamCalc` to calculate the cylindrical velocity components of `Urel` and `Uabs` (in `Allrun` script).
- Can be visualized in ParaView.

Multiple rotating frames of reference (MRF), theory

- Compute the absolute Cartesian velocity components, using the flux relative to the rotation of the local frame of reference (rotating or non-rotating)
- Development of the SRF equation, with convected velocity in the inertial reference frame (laminar version):

$$\nabla \cdot (\vec{u}_R \otimes \vec{u}_I) + \vec{\Omega} \times \vec{u}_I = -\nabla(p/\rho) + \nu \nabla \cdot \nabla(\vec{u}_I)$$

$$\nabla \cdot \vec{u}_I = 0$$

- The same equations apply in all regions, with different Ω.
  If $\vec{\Omega} = \vec{0}$, $\vec{u}_R = \vec{u}_I$
  Derivation at: `http://openfoamwiki.net/index.php/See_the_MRF_development`
- I.e., we need rotation in different regions, and compute absolute velocity.

The MRFSimpleFoam axialTurbine tutorials

- Run the axialTurbine_MRF_ggi tutorial:

```
cp -r $FOAM_TUTORIALS/incompressible/MRFSimpleFoam/axialTurbine_MRF_ggi \
    $FOAM_RUN
cd $FOAM_RUN/axialTurbine_MRF_ggi
./Allrun >& log_Allrun &
```

- Run the axialTurbine_MRF_mixingPlane tutorial:

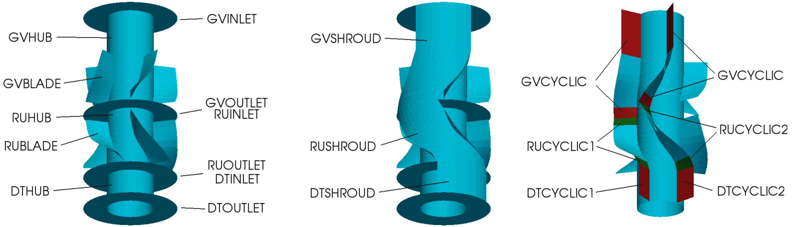
```
tut
cp -r incompressible/MRFSimpleFoam/axialTurbine_MRF_mixingPlane $FOAM_RUN
cd $FOAM_RUN/axialTurbine_MRF_mixingPlane
./Allrun >& log_Allrun &
```

- Look at the results:

```
paraFoam -builtin
```

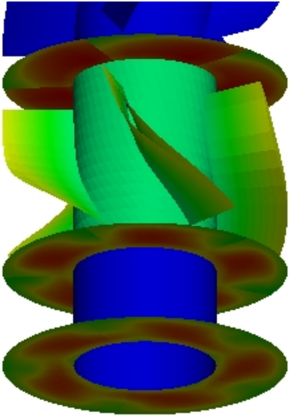
Use `Angular Periodic Filter` to copy to all blade passages (mark `Block Indices`, set `Rotation Angle` to 72, set `Axis` to Z)

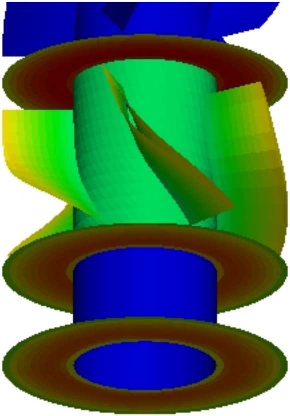
MRFSimpleFoam axialTurbine tutorial boundary names



- `GVOUTLET/RUINLET` and `RUOUTLET/DTINLET` coupled using `ggi/mixingPlane`.
- `GVCYCLIC` uses the regular cyclic boundary condition (discussed later)
- `{RU,DT}CYCLIC{1,2}` use the cyclicGgi boundary condition, as before

MRFSimpleFoam axialTurbine tutorial results





Note that the GGI solution resembles a snap-shot of a specific rotor orientation. Wakes will become unphysical!

The Allrun script (main features)

- Mesh generation (including rotor cellZone)
- Create GGI zones (necessary for parallel simulations)
- Run `MRFSimpleFoam` solver (using solver settings)
- Post-process (e.g. calculate cylindrical velocity components and `turboPerformance` engineering quantities)

Mesh generation (with rotor cellZone) and GGI zones

- Mesh generation as for SRF, with addition of `cellZone` for rotor. In `blockMeshDict`:

  ```
  hex (16 17 19 18 24 25 27 26)
  rotor
  (13 2 13)
  simpleGrading (1 1 1)
  ```

  Recommended to verify by visualizing the cell set/zone in ParaView (in `Allrun`: `foamToVTK -cellSet rotor`).
- GGI zones as for SRF, but now several more GGIs (see `setBatchGgi` and `decomposeParDict`, and visualize face zones in ParaView).

Alternative creation of cellZones

We need cellZones, which can be created e.g.

- directly in `blockMesh`
- from a multi-region mesh using `regionCellSets` and `setsToZones -noFlipMap`
- using the `cellSet` utility, the `cylinderToCell` cellSource, and `setsToZones -noFlipMap`
- in a third-party mesh generator, and converted using `fluent3DMeshToFoam`

Solver settings: MRFZones

For each zone in `constant/polyMesh/cellZones` (here only `rotor`):

```
rotor // Name of MRF zone
{
    origin    origin [0 1 0 0 0 0 0]  (0 0 0);
    axis      axis   [0 0 0 0 0 0 0]  (0 0 1);
    omega     omega  [0 0 -1 0 0 0 0] -10; //In radians per second

    // Walls that should not have the MRF zone solid-body velocity,
    // which is the default, but instead get the velocity from the
    // boundary condition, and all GGI interfaces in rotating zones:
    nonRotatingPatches ( RUSHROUD RUINLET RUOUTLET RUCYCLIC1 RUCYCLIC2 );
        //The shroud does not rotate, so absolute velocity should be zero.
        //Note that RUBLADE and RUHUB rotate although their
        //velocity is set to zero in the 0-directory (over-ridden)!
}
```

The addition of GGI patches to `nonRotatingPatches` was recent!

Solver settings: Inlet velocity boundary condition

- Swirling inlet velocity (here non-swirling):

```
GVINLET
{
    type                swirlInletVelocity;
    axis                (0 0 1);
    origin              (0 0 0);
    axialVelocity       constant -1;
    radialVelocity      constant 0;
    tangentialVelocity  constant 0;
    value               uniform (0 0 0);
}
```

Solver settings: Cyclic boundary condition for non-planar patches

- Only for conformal patches, and strict numbering requirement!
- In `constant/boundary` file:

```
GVCYCLIC
{
    type            cyclic;
    nFaces          240;
    startFace       11940;
    featureCos      0.9;
    transform       rotational;
    rotationAxis    (0 0 1);
    rotationCentre  (0 0 0);
    rotationAngle   -72; //Degrees from second half to first half
}
```

- Can of course also be used for planar patches.

Solver settings: The cyclic boundary condition - the field files

- The field files (`epsilon`, `k`, `nut`, `p`, `U`):

```
GVOUTLET
{
    type            cyclic;
}
RUINLET
{
    type            cyclic;
}
RUOUTLET
{
    type            cyclic;
}
DTINLET
{
    type            cyclic;
}
```

- Check your case set-up ...

Solver settings: Check setup of Cyclic boundary condition

- Check case set-up by modifying the cyclic debug switch (in `Allrun`):
  `runApplication $application -DebugSwitches cyclic=1`

  In log-file:

```
cyclicPolyPatch::calcTransforms : Writing half0 faces to file "...VTK/GVCYCLIC_half0_faces"
cyclicPolyPatch::calcTransforms : Writing half1 faces to file "...VTK/GVCYCLIC_half1_faces"
Prescribed transform: rotational.  Calculating transforms using Rodrigues Rotation.
    Axis = (0 0 1) centre = (0 0 0) angle = -72
cyclicPolyPatch::calcTransforms : Writing transform_half0 faces to file
    "...VTK/GVCYCLIC_transform_half0_faces"
```

- Visualize `*half{0,1}_faces` in ParaView.
  The `*transform_half0_faces` should end up at `*half1_faces`.

Solver settings: The mixingPlane interface - the boundary file

- For patches `GVOUTLET` and `RUINLET`:

```
GVOUTLET
{
    type            mixingPlane;
    nFaces          100;
    startFace       11840;
    shadowPatch     RUINLET;
    zone            GVOUTLETZone;
    coordinateSystem
    {
        type            cylindrical;
        origin          (0 0 0);
        axis            (0 0 1);
        direction       (1 0 0);
    }
    ribbonPatch
    {
        sweepAxis       Theta;
        stackAxis       R;
        followPatchDiscretisationParams
        {
            follow          bothPatches;
        }
    }
}
//RUINLET: vice versa, but does not need coordinateSystem and ribbonPatch,
//since it is slave (not the first of the pair)}
```

- `blockMesh` modifies the above to something equivalent but less readable.

Solver settings: The mixingPlane interface - the boundary file

Alternative `ribbonPatch` discretizations:

- ```
  discretisation followPatchDiscretisation;
  followPatchDiscretisationParams
  {
     follow masterPatch;
     //follow slavePatch;
  }
  ```
- ```
  discretisation  linearAlongStackAxis;
  linearAlongStackAxisParams
  {
      nRibbons 30;
  }
  ```
- ```
  discretisation  userDefinedProfile;
  userDefinedProfileParams
  {
      fileName            "ribbonPoints_localCoords_GVOUTLET_RUINLET.csv";
      nHeaderLine         3;
      componentColumns    (0 1 2);
      mergeSeparators     no;
      separator           ",";
      refColumn           -1;
      coordsType          local;
  }
  ```

  See example csv file in tutorial.
  ```

Solver settings: mixingPlane field files, fvSchemes and controlDict

- The field files (`epsilon, k, nut, p, U`):

```
GVOUTLET, RUINLET, RUOUTLET, DTINLET
{
    type            mixingPlane;
}
```

- The `mixingPlane` discretized in `fvSchemes`:

```
mixingPlane
{
    default         areaAveraging;
    U               areaAveraging;
    p               areaAveraging;
    k               fluxAveraging; //Transported variable
    epsilon         fluxAveraging; //Transported variable
    //NOTE: Ideally, tangential velocity components should also be
    //fluxAveraged, while keeping areaAveraging for normal velocity
    //component (to preserve perfect mass conservation)
}
```

Solver settings: Check mixingPlane setup

Activate `DebugSwitch`:

- In `Allrun`:

  `runApplication $application -DebugSwitches mixingPlane=2`

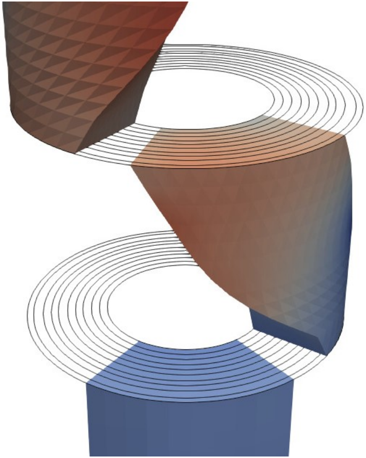
- In log-file:

```
Writing mixing plane patches in local coordinates as VTK. Master: GVOUTLET Shadow: RUINLET
Writing master and shadow raw profile points as VTK in global coordinates. Master: GVOUTLET Shadow: RUIN
Writing master and shadow raw profile points as CSV in global coordinates. Master: GVOUTLET Shadow: RUIN
Writing master and shadow raw profile points as VTK in local coordinates. Master: GVOUTLET Shadow: RUINL
Writing master and shadow raw profile points as CSV in local coordinates. Master: GVOUTLET Shadow: RUINL
Writing mixingPlane profile as VTK in global coordinates. Master: GVOUTLET Shadow: RUINLET
Writing mixingPlane profile as CSV in global coordinates. Master: GVOUTLET Shadow: RUINLET
Writing mixingPlane profile as VTK in local coordinates. Master: GVOUTLET Shadow: RUINLET
Writing mixingPlane profile as CSV in local coordinates. Master: GVOUTLET Shadow: RUINLET
Writing mixingPlane ribbons patch feature edges as VTK. Master: GVOUTLET Shadow: RUINLET
```

- Read VTK files into ParaView ...

Solver settings: Check mixingPlane setup

Visualization of `VTK/*ribbonPatchFeatureEdges.vtk`:

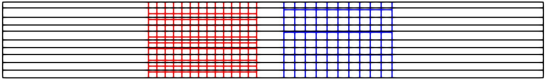


Solver settings: Check mixingPlane setup

Visualization of:

```
VTK/mixingPlane_*_localCoords_masterPatch*
VTK/mixingPlane_*_localCoords_ribbonPatch*
VTK/mixingPlane_*_localCoords_shadowPatch*
```

- Load all at the same time in ParaView. Rescale the y-component, since it is in angles or radians (angle: $-\pi \leq \theta \leq \pi$ or $-180 \leq \theta \leq 180$). For `axialTurbine_mixingPlane`, scale y by 0.001.



- Make sure that they overlap as they should.
- Make sure that they are flat enough (no wrinkles or wavyness).
- Make sure that the ribbons resolve both sides of the interface.

Solver settings: The mixingPlaneCheck functionObject

- Prints out the flux through `mixingPlane` interface pairs
- Entry in the `system/controlDict` file:

```
functions
(
    mixingPlaneCheck
    {
        // Type of functionObject
        type mixingPlaneCheck;

        phi                        phi;

        // Where to load it from (if not already in solver)
        functionObjectLibs ("libcheckFunctionObjects.so");
    }
);
```

- Output example (in log-file):

```
Mixing plane pair (GVOUTLET, RUINLET) : 0.0047 -0.00469787 Diff = 2.12159e-06 or 0.0451403 %
Mixing plane pair (RUOUTLET, DTINLET) : 0.00470227 -0.00470266 Diff = -3.88495e-07 or 0.00826186 %
```

Solver settings: The GGI interface - the boundary file

- For patches GVOUTLET and RUINLET:

```
GVOUTLET
{
    type            ggi;
    nFaces          100;
    startFace       11840;
    shadowPatch     RUINLET;
    zone            GVOUTLETZone;
    bridgeOverlap   false;
}
//RUINLET: vice versa
```

- The bridgeOverlap false setting does not allow geometrically non-matching patches. Always keep it like this unless you know what you are doing!
- The field files (epsilon, k, nut, p, U):

```
GVOUTLET, RUINLET, RUOUTLET, DTINLET
{
    type            ggi;
}
```

Post-processing: turboPerformance

- Add `functionObject` in `controlDict`.
- For details, see:
  https://openfoamwiki.net/index.php/Sig_Turbomachinery_Library_turboPerformance
- Example output:

```
Performance data:
   Head (m)    = 0.103277
   TOmega (W) = 3.22641
   Eff (%)     = 67.8919
   Forces      = (1.04595 -4.06676 -3.30422)
   Moments     = (-0.0681121 0.0725265 -0.322641)

 Fluid power output:
  dEm (W) = 4.75228
  Head (m) = 0.103277
```

- Directories: `turboPerformance` and `fluidPower`.

Special for MRF cases

- Note that the velocity, `U`, is the *absolute velocity*.
- At walls belonging to a rotational zone, that are not defined as `nonRotatingPatches`, the velocity boundary condition will be overridden and given a solid-body rotation velocity.
- The cell zones may be in multiple regions, as in the `axialTurbine` tutorials, and in a single region, as in the `mixerVessel2D` tutorial. GGI interfaces are thus not a requirement for MRF.
- **Always make sure that the interfaces between the zones are perfectly axi-symmetric**. Although the solver will probably run also if the mesh surface between the static and MRF zones is not perfectly symmetric about the axis, it will not make sense. Further, if a GGI is used at such an interface, continuity will not be fulfilled.

Dynamic Mesh (DyM), theory

- We will limit ourselves to non-deforming meshes with a fixed topology and a known rotating mesh motion
- Since the coordinate system remains fixed, and the Cartesian velocity components are used, the only change is the appearance of the relative velocity in convective terms. In cont. and mom. eqs.:

$$\int_S \rho\vec{v}\cdot\vec{n}dS \longrightarrow \int_S \rho(\vec{v}-\vec{v}_b)\cdot\vec{n}dS$$

$$\int_S \rho u_i\vec{v}\cdot\vec{n}dS \longrightarrow \int_S \rho u_i(\vec{v}-\vec{v}_b)\cdot\vec{n}dS$$

  where $\vec{v}_b$ is the integration boundary (face) velocity
- See derivation in:
  Ferziger and Perić, Computational Methods for Fluid Dynamics

The pimpleDyMFoam axialTurbine tutorial

- Run tutorial:

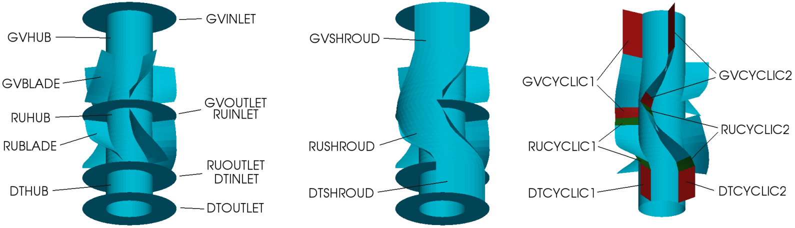
```
tut
cp -r incompressible/pimpleDyMFoam/axialTurbine_DyM_overlapGgi $FOAM_RUN
cd $FOAM_RUN/axialTurbine_DyM_overlapGgi
./Allrun >& log_Allrun &
```

- Have a look at the settings while the simulation is running...
- Look at the results (once the simulation is done):

```
paraFoam -builtin
```

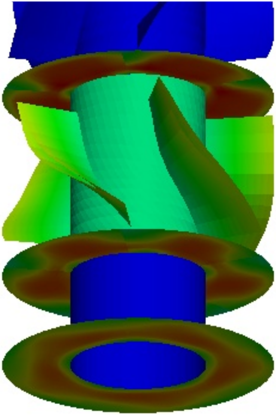
Use `Angular Periodic Filter` to copy to all blade passages (mark `Block Indices`, set `Rotation Angle` to 72, set `Axis` to Z)
Click on play!

pimpleDyMFoam axialTurbine tutorial boundary names (almost same as MRFSimpleFoam tutorials - GVCYCLIC differs)



- Runner region physically rotating
- `GVOUTLET/RUINLET` and `RUOUTLET/DTINLET` coupled by `overlapGgi`
- `overlapGgi` needs `patchConstrained` domain decomposition, see `system/decomposeParDict`

pimpleDyMFoam axialTurbine tutorial results



Note that wakes are now physical!

The Allrun script (main features)

- Mesh generation (<u>including rotor cellZone</u>)
- Create GGI zones (necessary for parallel simulations)
- Run `pimpleDyMFoam` solver (using solver settings)
- Post-process (e.g. calculate cylindrical velocity components and `turboPerformance` engineering quantities)

Mesh generation (with rotor cellZone) and GGI zones

- As for MRF

Solver settings: dynamicMeshDict

- In `constant/dynamicMeshDict`:

```
dynamicFvMesh       turboFvMesh;
turboFvMeshCoeffs
{
    coordinateSystem
    {
        type            cylindrical;
        origin          (0 0 0);
        axis            (0 0 1);
        direction       (1 0 0);
    }
    rpm { rotor  -95.49578; } //List rotating cellZones and rpm
    slider //List rotating coupled faceZones (needed for parallel simulations)
    {
        RUINLETZone -95.49578;
        RUOUTLETZone -95.49578;
        RUCYCLIC1Zone -95.49578;
        RUCYCLIC2Zone -95.49578;
    }
}
```

Solver settings: The profile1DfixedValue boundary condition

- For velocity (see case files for `k` and `epsilon`):

```
GVINLET
{
    type              profile1DFixedValue;
    fileName          "axialTurbineInletBC.csv";
    fileFormat        "turboCSV";
    interpolateCoord "R";
    fieldName         "Velocity";
    fieldScaleFactor 1;
    value             uniform (0 0 -1);
}
```

- In `constant/axialTurbineInletBC.csv`:

```
[Data]
 R [ m ], Velocity Axial [ m s^-1 ], Velocity Radial [ m s^-1 ], Velocity Circumferential [ m s^-1 ], Pr
0.05, -1, 0, 0, 0, 0.375, 14.855
0.075, -1, 0, 0, 0, 0.375, 14.855
0.1, -1, 0, 0, 0, 0.375, 14.855
```

Solver settings: Rotating (moving) patches

- Rotating patches:

```
RUBLADE
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
RUHUB
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
```

Solver settings: overlapGgi (boundary file and field files)

- For patches GVOUTLET and RUINLET:

```
GVOUTLET
{
    type            overlapGgi;
    nFaces          100;
    startFace       11840;
    rotationAxis    (0 0 1);
    nCopies         5;
    shadowPatch     RUINLET;
    zone            GVOUTLETZone;
}
```

```
RUINLET
{
    type            overlapGgi;
    nFaces          100;
    startFace       12820;
    rotationAxis    (0 0 1);
    nCopies         5;
    shadowPatch     GVOUTLET;
    zone            RUINLETZone;
}
```

  Note that nCopies must be the same on both sides!!!
- The field files (epsilon, k, nut, p, U):

```
GVOUTLET, RUINLET, RUOUTLET, DTINLET
{
    type            overlapGgi;
}
```

Post-processing: Visualize in ParaView etc.

- Remember to visualize:

  `paraFoam -builtin`

  Use `Angular Periodic Filter` to copy to all blade passages (mark `Block Indices`, set `Rotation Angle` to 72, set `Axis` to Z)
  Click on play!
- Check output of `turboPerformance`
- Check output of `ggiCheck` and
  `-DebugSwitches GGIInterpolation=1`

Questions?

Further information

- https://sourceforge.net/projects/turbowg/
- http://openfoamwiki.net/index.php/Sig_Turbomachinery
- http://www.tfd.chalmers.se/~hani/kurser/OS_CFD
- http://www.abcfd.se