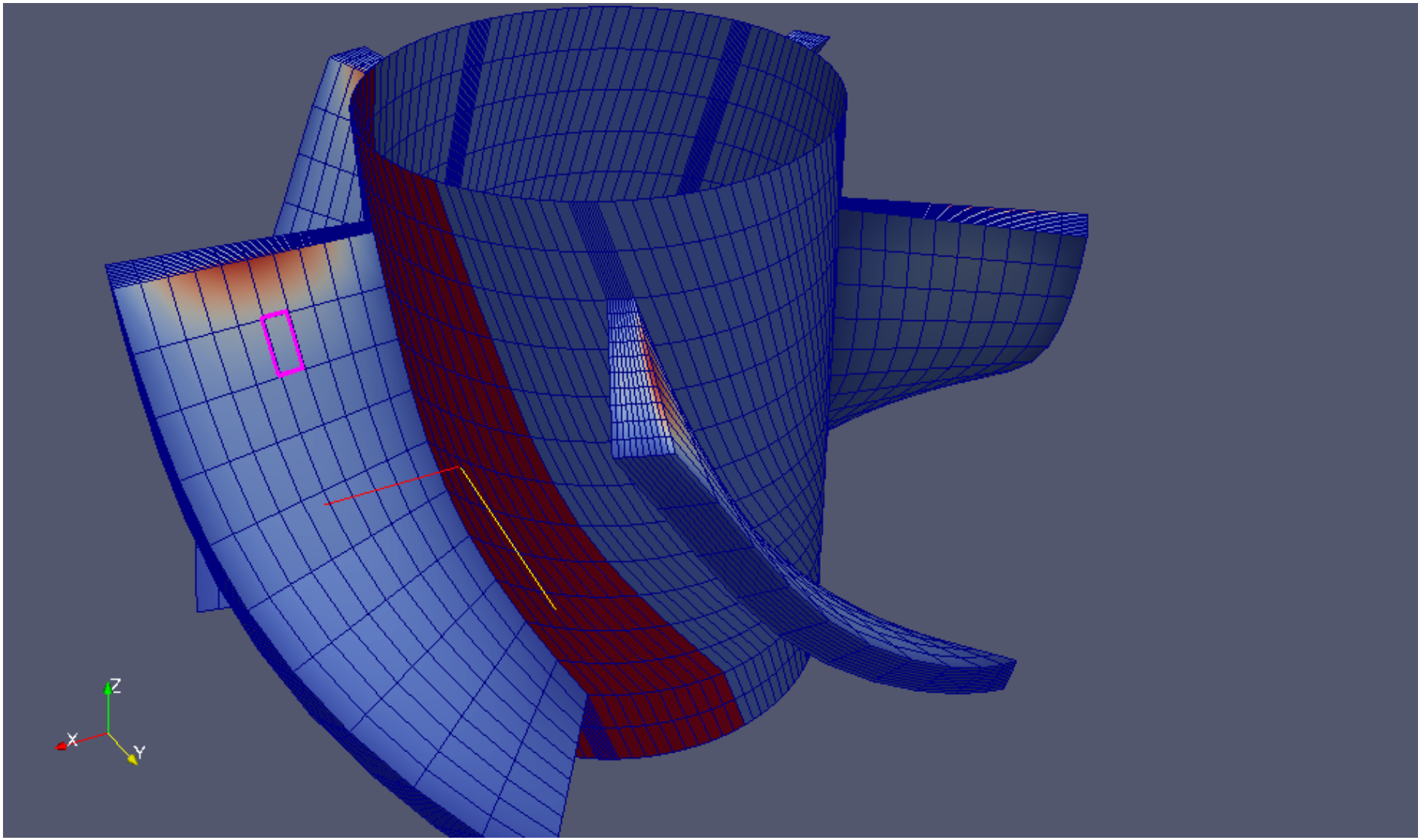
Strongly coupled FSI analysis on the axialTurbine tutorial case for the course CFD with OpenSource Software

Erik Karlsson

2014-12-01

Presentation outline

- fsiFoam
- Important files
- Modifications
- Running the case
- Possible improvements/left to do


Z
X
Y

fsiFoam

- fsiFoam is a partitioned strongly coupled FSI FVM solver.
  - A partitioned method solves the the governing equations of the flow and structure separately, with two independent solvers.
  - Strongly coupled implies an implicit coupling between fluid and solid

fsiFoam

```
for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;
        fsi.initializeFields();
        fsi.updateInterpolator();
        scalar residualNorm = 0;

        do
        {
            fsi.outerCorr()++;
            fsi.updateDisplacement(); // Using selected coupling scheme
            fsi.moveFluidMesh();
            fsi.flow().evolve();
            fsi.updateForce();       // Face Ggi interpolation
            fsi.stress().evolve();

            residualNorm =
                fsi.updateResidual(); // Point Ggi interpolation
        }
        while
        (
            (residualNorm > fsi.outerCorrTolerance())
         && (fsi.outerCorr() < fsi.nOuterCorr())
        );
}
```

fsiFoam

- General algorithm for fsiFoam
  1. Update fluid and structural mesh
  2. Fluid solver
  3. Solid solver
  4. Compute residual
- Iterate 1-4 until residual$<$tol
  5. Next time step.

Getting started

- Case files can be downloaded at pingpong.
  Extract by and move by:

```
cd *downloadDir*
tar -xzvf axialTurbineFsiTut.tar.gz
rm axialTurbineFsiTut.tar.gz
run
mv *downloadDir*/axialTurbineFsiTut/ .
cd axialTurbineFsiTut
```

Fsi package

The FSI package is not included in Foam 3.1-extend, it has to be downloaded:

`http://openfoamwiki.net/index.php/Extend-bazaar/Toolkits/Fluid-structure_interaction`

Then extract and install by:

```
run
mv *downloadDir*/Fsi_31.tar.gz .
tar -xzvf Fsi_31.tar.gz
rm Fsi_31.tar.gz
cd FluidStructureInteraction/
cd src/
./Allmake
```

Important files

- A look at the axialTurbineFsiTut case

```
cd $FOAM_RUN/axialTurbineFsiTut
ls
fluid  makeLinks makeSerialLinks removeSerialLinks solid
```

- `fluid`
  Contains all the files for the fluid computation
- `solid`
  Contains files for the solid computation.
- The make/removeLinks will link the two computations.

Fluid

- `cd fluid/constant`

  `dynamicMeshDict flowProperties fsiProperties transportProperties`
- `$vim dynamivMeshDict`

  `dynamicFvMesh dynamicMotionSolverFvMesh;`

  ```
  solver velocityLaplacian;
  diffusivity quadratic inverseDistance 1(RUBLADE);
  ```
- Dynamic mesh due to displacement of the fsi interface.

Fluid

- Settings for coupling the solvers

```
$vim fsiProperties
 solidPatch RUBLADEsolid;    //Solid interface patch
 solidZone  RUBLADEsolidZone;
 fluidPatch RUBLADE;     //Fluid interface patch
 fluidZone  RUBLADEZone;
 relaxationFactor 0.10;
 interfaceDeformationLimit 0; //Mesh update limit
 outerCorrTolerance 1e-6;
 nOuterCorr 30;
 interpolatorUpdateFrequency 0;
 couplingScheme Aitken;
 couplingReuse 0;
 coupled yes;
```

Fluid

- The coupling process requires information of FSI zones
  These are specified in `axialTurbineFsiTut/fluid/setBatchGgi`
  and then created in the `Allrun` script.
- Zones for the Ggi interpolation is also created.
- The fsi zone for the solid part is created in an similar fashion

Solid

- And now a look at the files for the solid computation.
- `cd solid/constant`
  `rheologyProperties stressProperties`
- `$vim rheologyProperties`

```
planeStress no; //No for 3D, yes for 2D
 rheology
 {
     type linearElastic;
     rho rho [1 -3 0 0 0 0 0] 1000; // Density
     E E [1 -1 -2 0 0 0 0] 5.0e6;   // Young's modulus
     nu nu [0 0 0 0 0 0 0] 0.4;   // Poisson's ratio
 }
```

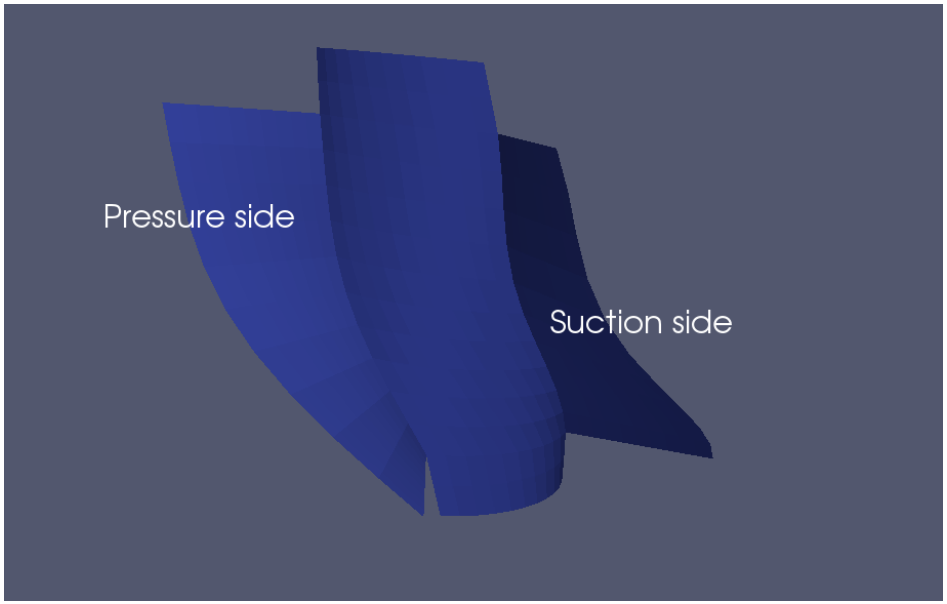
Solid

- $vim stressProperties

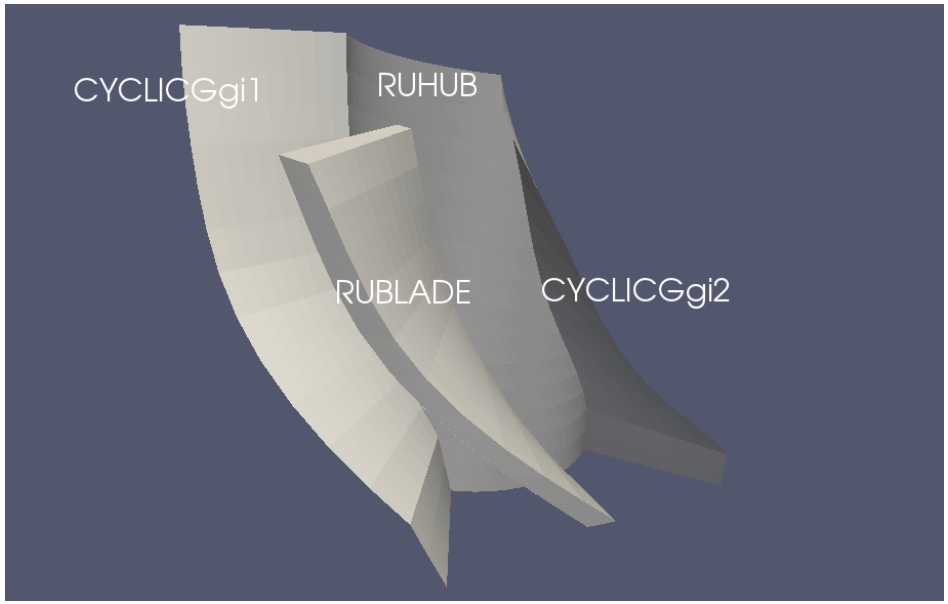
```
stressModel unsTotalLagrangianStress;
  unsTotalLagrangianStressCoeffs
  {
      nCorrectors 1000;
      convergenceTolerance 1e-7;
      relConvergenceTolerance 1e-3;
      nonLinear yes;
      debug no;
      moveMesh yes;
  }
```

- Here the stress model is specified. `unsTotalLagrangianStress` is a large strain, elastic stress analysis solver based on total Lagrangian displacement formulation.

Modifications



Original geometry



Modified geometry

The blockMeshDict

- The geometry is based on the `axialTurbine` in `tutorials/incompressible/simpleSRFFoam/axialTurbine/`
- Copy the geometry in order to make the necessary modifications

```
run
cd axialTurbineFsiTut/fluid/constant/polyMesh/
rm blockMeshDict.m4
cp $FOAM_TUTORIALS/incompressible/simpleSRFFoam/
axialTurbine/constant/polyMesh/blockMeshDict.m4 .
```

The blockMeshDict

- The `blockMeshDict` is created with `m4`.
- A look in the `m4` file:
  `vim blockMeshDict.m4`

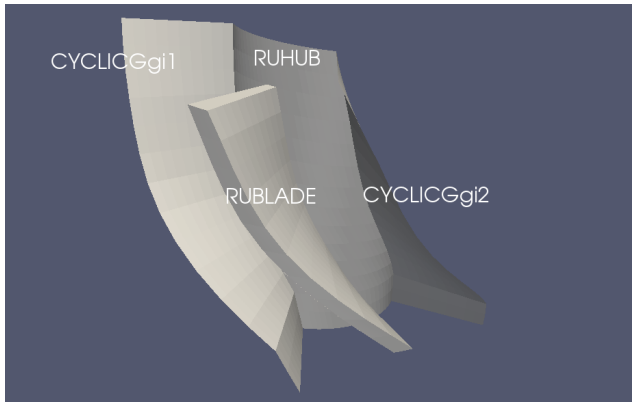
The blockMeshDict

- First some general macros to create 2D/ extruded 2D-meshes

```
changecom(//)changequote([,])
define(calc, [esyscmd(perl -e 'printf ($1)')])
define(VCOUNT, 0)
define(vlabel, [[// ]Vertex $1 = VCOUNT define($1,
VCOUNT)define([VCOUNT],incr(VCOUNT))])

define(pi, calc(3.14159265/20)) // Note how pi is defined
define(hex2D, hex ($1b $2b $3b $4b $1t $2t $3t $4t))
define(quad2D, ($1b $2b $2t $1t))
define(frontQuad, ($1t $2t $3t $4t))
define(backQuad, ($1b $4b $3b $2b))
```

The blockMeshDict

```
define(hr, 0.05)// Hub radius (m)
define(sr, 0.1)// Shroud radius (m)
define(h, 0.005/hr/20)// Blade thickness (m)
// RUNNER REGION GEOMETRY AND MESH PROPERTIES
define(RUial, 0.02)// Runner inlet axial length (m)
define(RUal, 0.1)// Runner axial length (m)
define(RUoal, 0.02)// Runner outlet axial length (m)
define(RUnb, 5)//Runner blades per 360 degrees (integer!)
define(RUrc, 10)//Cells in radial direction in runner
define(RUtc,10)//Cells in tangential direction between blades
define(RUiac, 2)// Cells in axial direction at inlet
define(RUbac, 10) //Cells in axial direction between blades
define(RUoac, 2)// Cells in axial direction at outlet
```

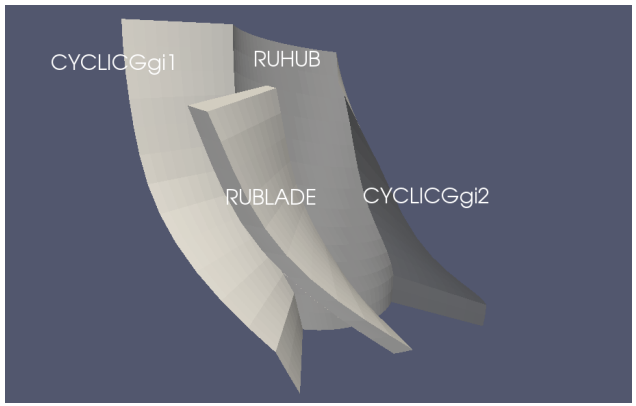


The blockMeshDict

```
// TANGENTIAL PITCHES (RADIANS)
define(RUp, calc(2*pi/RUnb))
// TANGENTIAL SHIFTS BETWEEN AXIAL LEVELS (BOTTOM-UP)
define(RUts01, calc(-1/10*RUp))//Shift from level RU0 to RU1
define(RUts12, calc(-4/5*RUp))//Shift from level RU1 to RU2
define(RUts23, calc(-1/10*RUp))//Shift from level RU2 to RU3
//AXIAL/TANGENTIAL BASE POINTS FOR EACH LEVEL (BOTTOM-UP)
//(CENTER OF RUNNER SET TO THETA=0, Z=0)
define(RUa0, calc(-RUoal-0.5*RUal)) //Center runner
define(RUt0, calc(-0.5*RUp-(0.5*RUts12))) //Center runner
define(RUt1, calc(RUt0+RUts01))
define(RUt2, calc(RUt1+RUts12))
define(RUt3, calc(RUt2+RUts23))
```

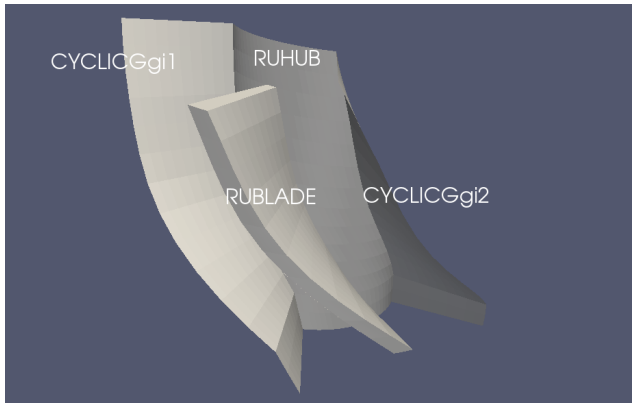
The blockMeshDict

```
 vertices //(radial [m], tangential [radians], axial [m])
(
//Runner hub:
(hr calc(RUt0+0.5*RUp) RUa0) vlabel(RU0lb)
(hr calc(RUt0+RUp-h) RUa0) vlabel(RU0rb)
(hr calc(RUt1+0.5*RUp) calc(RUa0+RUoal)) vlabel(RU1lb)
(hr calc(RUt1+RUp-h) calc(RUa0+RUoal)) vlabel(RU1rb)
(hr calc(RUt2+0.5*RUp) calc(RUa0+RUoal+RUal)) vlabel(RU2lb)
(hr calc(RUt2+RUp-h) calc(RUa0+RUoal+RUal)) vlabel(RU2rb)
(hr calc(RUt3+0.5*RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(RU3lb)
(hr calc(RUt3+RUp-h) calc(RUa0+RUoal+RUal+RUial)) vlabel(RU3rb)
```



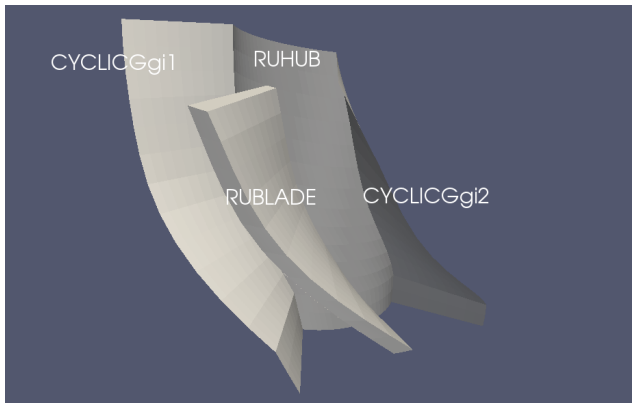
The blockMeshDict

```
// extras
(hr calc(RUt0+RUp) RUa0) vlabel(extraRU0rb)
(hr calc(RUt1+RUp) calc(RUa0+RUoal)) vlabel(extraRU1rb)
(sr calc(RUt0+RUp) RUa0) vlabel(extraRU0rt)
(sr calc(RUt1+RUp) calc(RUa0+RUoal)) vlabel(extraRU1rt)
(hr calc(RUt2+RUp) calc(RUa0+RUoal+RUal)) vlabel(extraRU2rb)
(hr calc(RUt3+RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(extraRU3rb)
(sr calc(RUt2+RUp) calc(RUa0+RUoal+RUal)) vlabel(extraRU2rt)
(sr calc(RUt3+RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(extraRU3rt)
```



The blockMeshDict

```
//rExtras
(hr calc(RUt0+1.5*RUp) RUa0) vlabel(rExtraRU0rb)
(hr calc(RUt1+1.5*RUp) calc(RUa0+RUoal)) vlabel(rExtraRU1rb)
(sr calc(RUt0+1.5*RUp) RUa0) vlabel(rExtraRU0rt)
(sr calc(RUt1+1.5*RUp) calc(RUa0+RUoal)) vlabel(rExtraRU1rt)
(hr calc(RUt2+1.5*RUp) calc(RUa0+RUoal+RUal)) vlabel(rExtraRU2rb)
(hr calc(RUt3+1.5*RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(rExtraRU3rb)
(sr calc(RUt2+1.5*RUp) calc(RUa0+RUoal+RUal)) vlabel(rExtraRU2rt)
(sr calc(RUt3+1.5*RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(rExtraRU3rt)
);
```



The blockMeshDict

```
//blocks
hex2D(RU0r, extraRU0r, extraRU1r, RU1r)
rotor
(RUtc RUoac RUrc)
simpleGrading (1 1 1)
hex2D(RU2r, extraRU2r,extraRU3r,RU3r)
rotor
(RUtc RUoac RUrc)
simpleGrading (1 1 1)
hex2D(extraRU0r, rExtraRU0r, rExtraRU1r, extraRU1r)
rotor
(RUtc RUoac RUrc)
simpleGrading (1 1 1)
hex2D(extraRU1r, rExtraRU1r, rExtraRU2r, extraRU2r)
rotor
(RUtc RUbac RUrc)
simpleGrading (1 1 1)
hex2D(extraRU2r, rExtraRU2r, rExtraRU3r, extraRU3r)
rotor
(RUtc RUiac RUrc)
simpleGrading (1 1 1)
```

The blockMeshDict

```
edges // Inappropriate with arc due to coordinate conversion
(
//Runner
    spline RU1lt RU2lt
    (
        (sr calc(RUt1+0.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal)
    )
    spline RU1lb RU2lb
    (
        (hr calc(RUt1+0.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal)
    )
    spline RU1rt RU2rt
    (
        (sr calc(RUt1+RUp+0.75*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
    spline RU1rb RU2rb
    (
        (hr calc(RUt1+RUp+0.75*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
```

The blockMeshDict

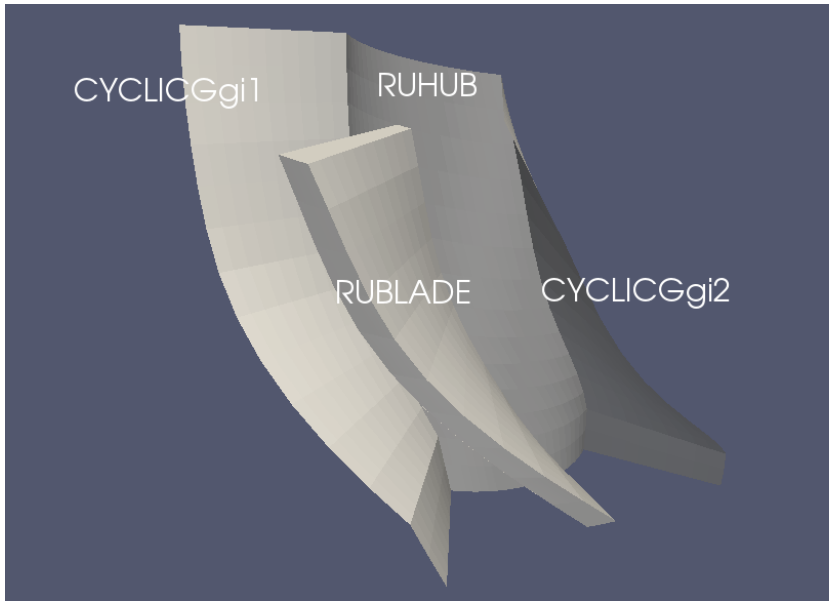
```
//extraSplines
    spline extraRU1rt extraRU2rt
    (
        (sr calc(RUt1+RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
    spline extraRU1rb extraRU2rb
    (
        (hr calc(RUt1+RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
    spline rExtraRU1rt rExtraRU2rt
    (
        (sr calc(RUt1+1.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal)
    )
    spline rExtraRU1rb rExtraRU2rb
    (
        (hr calc(RUt1+1.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal)
    )
);
```

The blockMeshDict

```
// boundary
RUINLET
{
    type            patch;
    faces
    (
        quad2D(RU3r, RU3l)
        quad2D(extraRU3r,RU3r)
        quad2D(rExtraRU3r, extraRU3r)
    );
}
```

The blockMeshDict

```
RUCYCLIC1
{
    type            cyclicGgi;
    shadowPatch     RUCYCLIC2;
    zone            RUCYCLIC1Zone;
    bridgeOverlap   false; //No uncovered faces
    rotationAxis    (0 0 1);
    rotationAngle   72; //Rotation angle to shadowPatch
    separationOffset (0 0 0);
    faces
    (
        quad2D(RU1l, RU0l)
        quad2D(RU2l,RU1l)
        quad2D(RU3l, RU2l)
    );
}
```



GGI

- GGI is used to couple interfaces in the mesh where the nodes on each side of the interface does not match. Weight factors are used to decide how much information should be transferred from one side of the ggi to its neighbouring cells on the other side. The use of ggi in our case is likely unnecessary but is used since it used in the original geometry.
- `cyclic` would probably be enough.

The blockMeshDict

```
RUBLADE
{
    type wall;
    faces
    (
        quad2D(RU1r, RU2r) //Blade left
        quad2D(extraRU1r, RU1r) //wall bottom
        quad2D(RU2r, extraRU2r) //Wall top
        quad2D(extraRU2r, extraRU1r) //Blade right
    );
}
```

Solid geometry

- Similarly done for the solid geometry.
  There is however only one patch - the solid fsi patch
- The `blockMeshDict` is created in the `Allrun` script

Special BC

- For the solid part we have a BC, `tractionDisplacement`, on the interaction interface.
  `run`
  `vim axialTurbineFsiTut/solid/0/D`

```
RUBLADEsolid
{
    type            tractionDisplacement;
    traction        uniform (0 0 0);//External traction
    pressure        uniform 0;      //External pressure
    value           uniform (0 0 0);//External displacement
}
```

Special BC

- For the fluid fsi interface vi have the BC, `movingWallVelocity`

  `run`
  `vim axialTurbineFsiTut/fluid/0_orig/U`

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

- This BC corrects the flux due to mesh motion in such a way that the total flux through the moving wall is zero.

Running the case

- First we have to create the mesh and fsi zones for the Solid part
  This is done by running the `Allrun` script

  ```
  run
  cd axialTurbineFsiTut/solid
  ./Allrun
  ```

- Next create the links

  ```
  cd ..
  ./makeSerialLinks fluid/ solid/
  ```

- Then create mesh and zones for the fluid.

  ```
  cd fluid
  ./Allrun
  ```

- Now we can run the case with:
  `fsiFoam >&log.fsiFoam &`

Allrun script

- The `Allrun` script

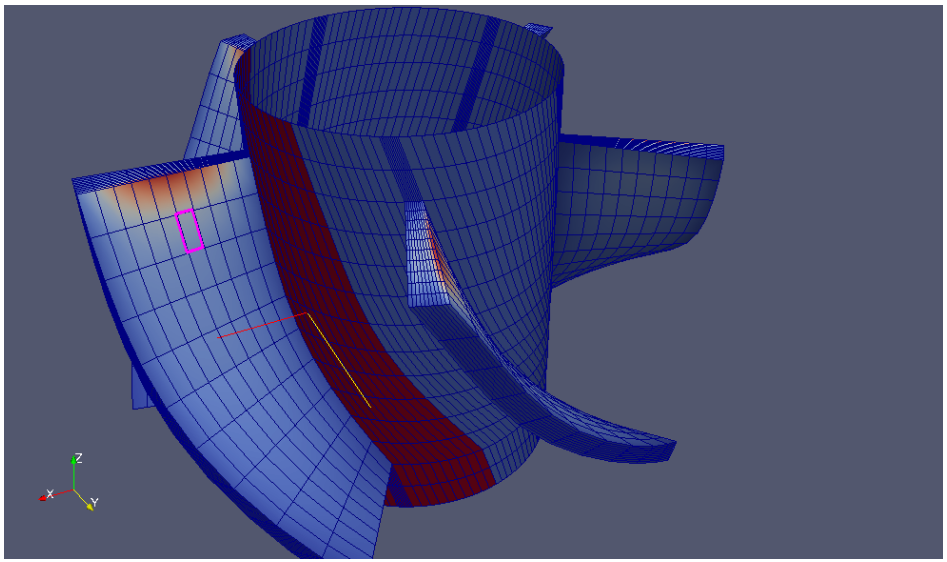
```
#!/bin/bash
# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# Get application from system/controlDict
application=‘getApplication‘

m4 < constant/polyMesh/blockMeshDict.m4 > constant/polyMesh/blockMeshDict
runApplication blockMesh
transformPoints -scale "(1 20 1)"
transformPoints -cylToCart "((0 0 0) (0 0 1) (1 0 0))"

# Create GGI, fsi set:
runApplication setSet -batch setBatchGgi
runApplication setsToZones -noFlipMap

#runApplication $application
```

setBatchGgi

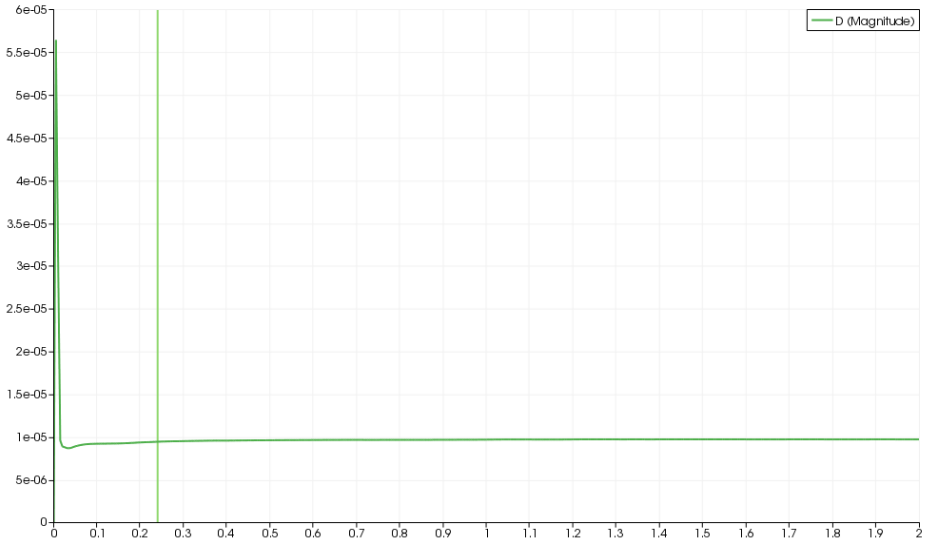
- In `axialTurbineFsiTut/fluid` we have a file, `setBatchGgi`

```
faceSet RUCYCLIC1Zone new patchToFace RUCYCLIC1
faceSet RUCYCLIC2Zone new patchToFace RUCYCLIC2
faceSet RUBLADEZone new patchToFace RUBLADE
quit
```

Result



Isometric view, colored by deflection



Deflection over time

Post-processing

- To copy the geometry and rotate the copy around the z-axis a filter, `transform`, is used.
- The filter is used five times the the rotation angles: 72, 144, -72 and -144 degrees.
- The state is then saved as `allBlades.pvsm` and can be loaded with `paraview --state=allBlades.pvsm`

Future work

- Improve the blade geometry
  - As it is defined now the thickness of the blade increases with radius
  - Add a layer of fluid at the shroud in order to let the "blade shroud" be a part of the fsi patch
- Add Coriolis terms in the flow model or rotating mesh
  - The flow is purely axial which makes the "turbine" more a nozzle stator

Thank you!

Questions?