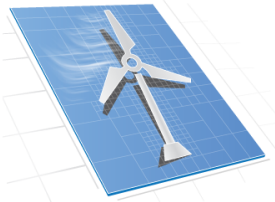
HyperCFD
Fertınaz Yazılım Ltd.

OpenFOAM Workshop
Teknopark Istanbul

06 Solvers & Postprocess
January 2017

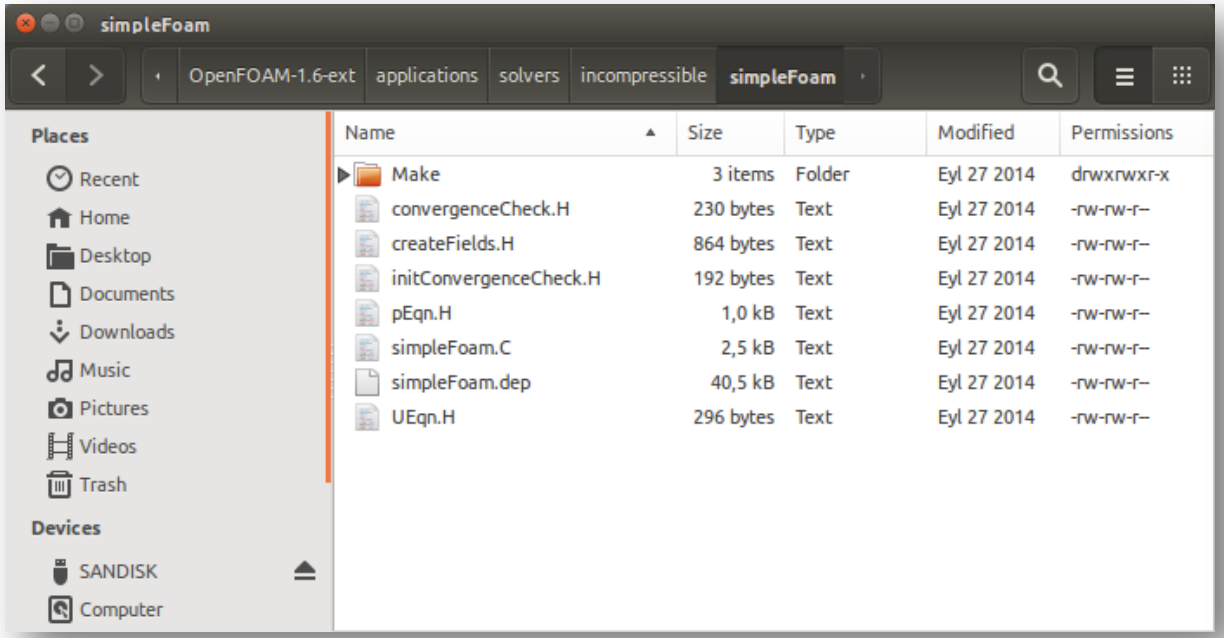
HyperCFD – Disclaimer

This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OpenFOAM and OpenCFD trademarks.

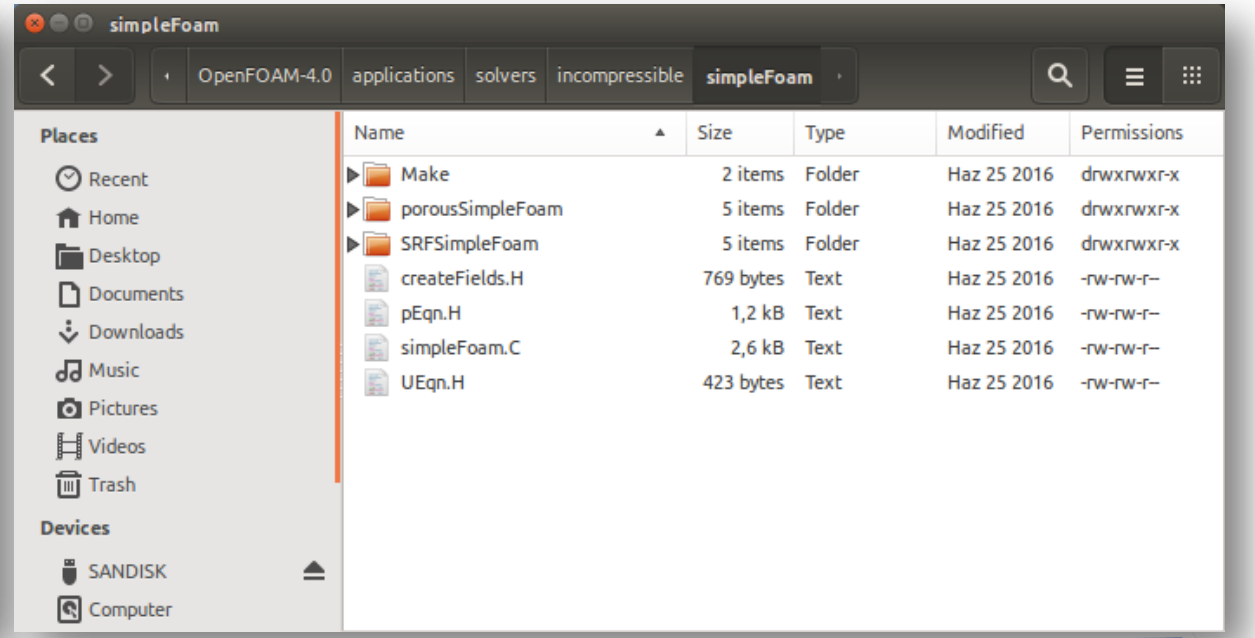


HyperCFD – OpenFOAM Flow Solvers

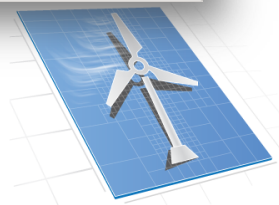
- List of solvers is shared
- This is how a solver source code directory looks like



- An older version of OpenFOAM (1.6-ext)



- A recent version of OpenFOAM (4.0)



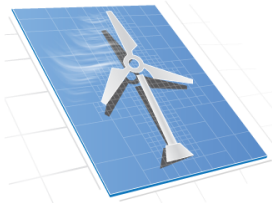
HyperCFD – OpenFOAM Flow Solvers

- A general OpenFOAM solver code consists of:
  - Top level code: Idea based on equation mimicking
  - Make files: Compilation files

```
fertinaz  staff   170 Jun 17  2014 Make
fertinaz  staff   296 Jun 17  2014 UEqn.H
fertinaz  staff   230 Jun 17  2014 convergenceCheck.H
fertinaz  staff   864 Jun 17  2014 createFields.H
fertinaz  staff   192 Jun 17  2014 initConvergenceCheck.H
fertinaz  staff  1015 Jun 17  2014 pEqn.H
fertinaz  staff  2455 Jun 17  2014 simpleFoam.C
```

```
fertinaz  staff   170 Jun 17  2014 Make
fertinaz  staff   706 Jun 17  2014 UEqn.H
fertinaz  staff  1401 Jun 17  2014 createFields.H
fertinaz  staff   374 Jun 17  2014 hEqn.H
fertinaz  staff  2356 Jun 17  2014 pEqn.H
fertinaz  staff  3018 Jun 17  2014 rhoPimpleFoam.C
```

```
// Solve the Momentum equation

tmp<fvVectorMatrix> UEqn
(
    fvm::div(phi, U)
  + turbulence->divDevReff(U)
);

UEqn().relax();

eqnResidual = solve
(
    UEqn() == -fvc::grad(p)
).initialResidual();
```



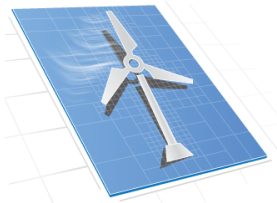
HyperCFD – OpenFOAM Flow Solvers

- Equation mimicking in OpenFOAM
- Lots of examples can be found in the internet and this one is from Jasak's presentations:
- Turbulent kinetic energy equation

$$k_t + \nabla \cdot (\mathbf{u}k) - \nabla \cdot ((\nu + \nu_t) \nabla k) = \nu_t (\tfrac{1}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^T)) - k (eps_0 / k_0)$$

can be represented as

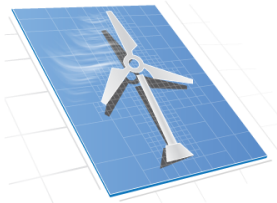
```
solve(
      fvm::ddt(k)
    + fvm::div(phi,k)
    - fvm::laplacian(nu() + nut, k)
   == nut*magSqr(symm(fvc::grad(U)))
    - fvm::Sp(epsilon/k, k)
);
```



HyperCFD – OpenFOAM Flow Solvers

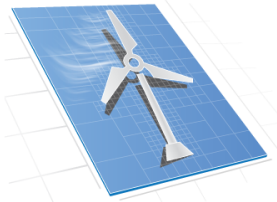
- Equation mimicking in OpenFOAM is implemented in the `UEqn.H, pEqn.H` files
- These are the top level code: Easy to follow, modify and understand
- If you would like to write a new solver
  - Do not start from scratch, start with copying the most similar solver for your custom need
  - Then change the file names – `simpleFoam.C` to `mysimpleFoam.C`
  - Then change these occurrences inside the files
  - Then build your code according to your mathematical model

```
tmp<fvVectorMatrix> UEqn
(
    fvm::div(phi, U)
  + turbulence->divDevReff(U)
);

UEqn().relax();

eqnResidual = solve
(
    UEqn() == -fvc::grad(p)
).initialResidual();
```



HyperCFD – OpenFOAM Flow Solvers

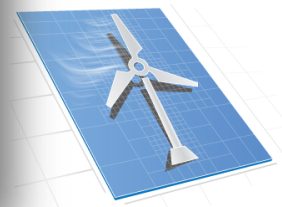
- We are ready to run a solver now
- Mesh is completed: `constant/polyMesh`
- IC and BC are in `0/`
- Turbulence is in `0/` and `constant/`
- Discretization schemes and solution methods are all entered: `system/fvSchemes` and `system/fvSolution`
- Suppose you need a incompressible, steady, turbulent solver
  - `simpleFoam`
  - Go to your case folder and type the solver name in the terminal to run it.



HyperCFD – OpenFOAM Flow Solvers

- When you start running the solver this is the typical output you see
- Settings are checked
  - Fields are created: U and p
  - Fluxes are generated
  - Convergence criteria checked
  - Turbulence model checked
  - MRF regions checked

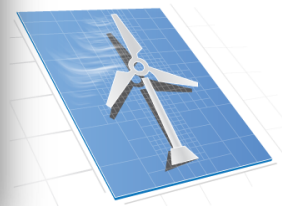
```
Build  : 4.0
Exec   : simpleFoam
Date   : Jan 15 2017
Time   : 18:50:39
Host   : "fertinaz-M4800"
PID    : 18673
Case   : /home/fertinaz/OpenFOAM/OpenFOAM-4.0/tutorials/incompressible/simpleFoam/airFoil2D
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0


SIMPLE: convergence criteria
    field p       tolerance 1e-05
    field U       tolerance 1e-05
    field nuTilda       tolerance 1e-05

Reading field p

Reading field U

Reading/calculating face flux field phi

Selecting incompressible transport model Newtonian
Selecting turbulence model type RAS
Selecting RAS turbulence model SpalartAllmaras
Selecting patchDistMethod meshWave
SpalartAllmarasCoeffs
{
    sigmaNut        0.66666;
    kappa           0.41;
    Cb1             0.1355;
    Cb2             0.622;
    Cw2             0.3;
    Cw3             2;
    Cv1             7.1;
    Cs              0.3;
}

No MRF models present

No finite volume options present
```



HyperCFD – OpenFOAM Flow Solvers

- Then the simulation starts
- You can follow residuals
  - pyFoam library
  - `pyFoamPlotWatcher.py`
- Also you can check CFL number if you're running a transient simulation
- Adding probes is another trick
- Use `functionObjects` to compute forces at the end of each iteration

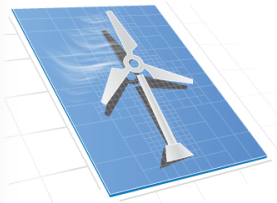
```
Starting time loop

Time = 1

smoothSolver:  Solving for Ux, Initial residual = 1, Final residual = 0.0611362, No Iterations 2
smoothSolver:  Solving for Uy, Initial residual = 1, Final residual = 0.0585177, No Iterations 2
GAMG:  Solving for p, Initial residual = 1, Final residual = 0.0940365, No Iterations 7
time step continuity errors : sum local = 0.000301165, global = 6.11097e-18, cumulative = 6.11097e-18
smoothSolver:  Solving for nuTilda, Initial residual = 1, Final residual = 0.0452916, No Iterations 2
ExecutionTime = 0.09 s  ClockTime = 0 s

Time = 2

smoothSolver:  Solving for Ux, Initial residual = 0.431056, Final residual = 0.01751, No Iterations 4
smoothSolver:  Solving for Uy, Initial residual = 0.547535, Final residual = 0.0165972, No Iterations 4
GAMG:  Solving for p, Initial residual = 0.75244, Final residual = 0.0588633, No Iterations 5
time step continuity errors : sum local = 0.000380151, global = 3.45666e-17, cumulative = 4.06776e-17
smoothSolver:  Solving for nuTilda, Initial residual = 0.138687, Final residual = 0.00977524, No Iterations 2
ExecutionTime = 0.11 s  ClockTime = 0 s

Time = 3

smoothSolver:  Solving for Ux, Initial residual = 0.174335, Final residual = 0.00728614, No Iterations 4
smoothSolver:  Solving for Uy, Initial residual = 0.157099, Final residual = 0.00587833, No Iterations 4
GAMG:  Solving for p, Initial residual = 0.737611, Final residual = 0.050711, No Iterations 7
time step continuity errors : sum local = 0.000386109, global = 1.55191e-17, cumulative = 5.61967e-17
smoothSolver:  Solving for nuTilda, Initial residual = 0.12836, Final residual = 0.00310622, No Iterations 4
ExecutionTime = 0.14 s  ClockTime = 0 s

Time = 4

smoothSolver:  Solving for Ux, Initial residual = 0.0620803, Final residual = 0.00495272, No Iterations 2
smoothSolver:  Solving for Uy, Initial residual = 0.058189, Final residual = 0.00236819, No Iterations 4
GAMG:  Solving for p, Initial residual = 0.682536, Final residual = 0.0565832, No Iterations 6
time step continuity errors : sum local = 0.000349528, global = 1.6746e-17, cumulative = 7.29428e-17
smoothSolver:  Solving for nuTilda, Initial residual = 0.171253, Final residual = 0.00520237, No Iterations 4
ExecutionTime = 0.16 s  ClockTime = 0 s

Time = 5

smoothSolver:  Solving for Ux, Initial residual = 0.140962, Final residual = 0.00563236, No Iterations 4
smoothSolver:  Solving for Uy, Initial residual = 0.151027, Final residual = 0.00436528, No Iterations 4
GAMG:  Solving for p, Initial residual = 0.79492, Final residual = 0.0333985, No Iterations 5
time step continuity errors : sum local = 0.000222125, global = 1.75263e-17, cumulative = 9.04691e-17
smoothSolver:  Solving for nuTilda, Initial residual = 0.119073, Final residual = 0.00342974, No Iterations 4
ExecutionTime = 0.18 s  ClockTime = 0 s

End
```



HyperCFD – OpenFOAM Flow Solvers

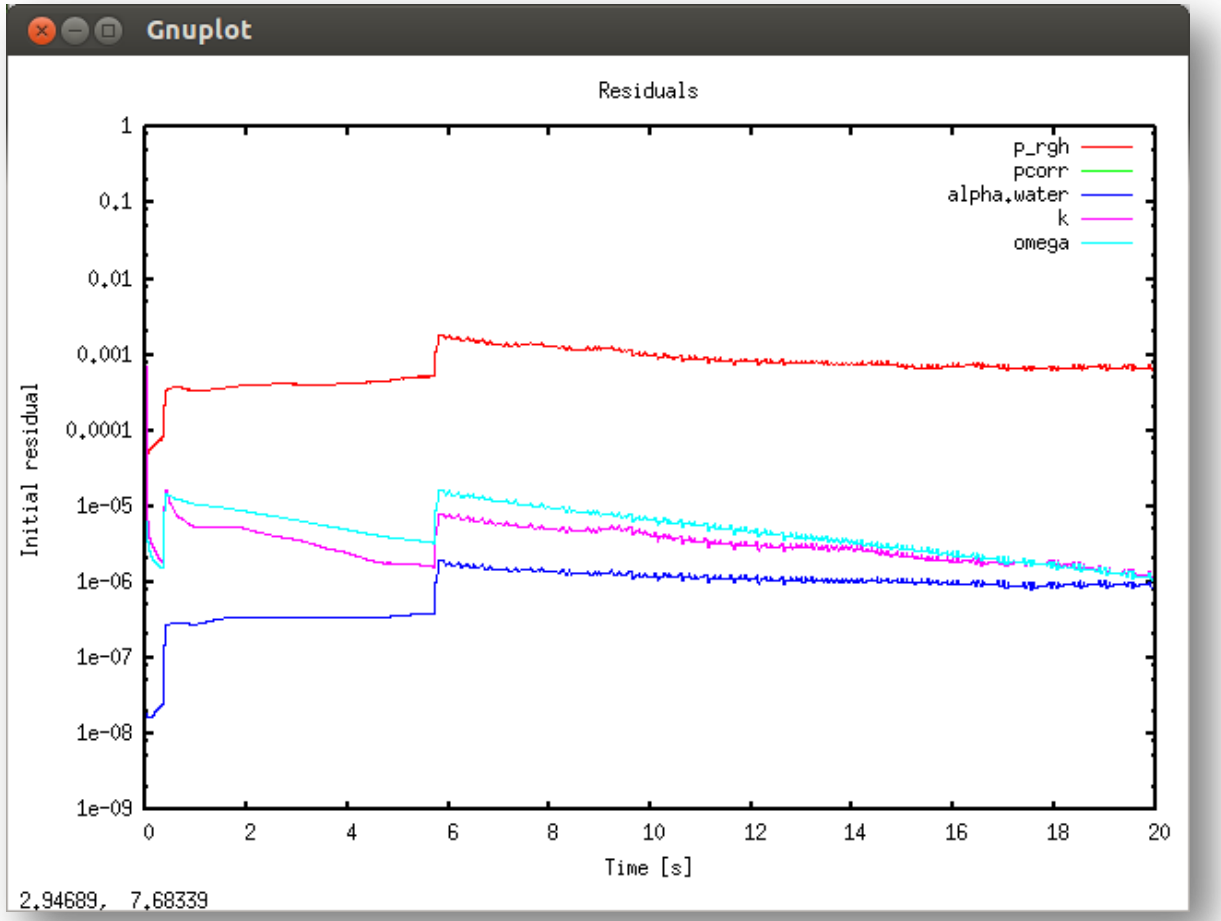
- Lets take a look at `system/controlDict`
  - `application`: Name of the solver
  - `startFrom`: Continue from latest time step or start from scratch
  - `stopAt`: `writeNow` or `endTime`
  - `deltaT`: Timestep
  - `writeInterval`: Number of steps between each file write
  - `purgeWrite`: Keep all directories or replace new ones with the existing ones

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application simpleFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1000;

deltaT          1;

writeControl    timeStep;

writeInterval   50;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

// ************************************************************************* //
```



HyperCFD – OpenFOAM Post-Processing

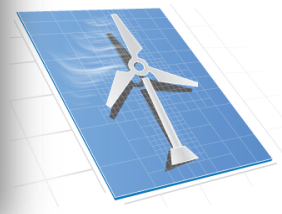
- Lets take a look at residuals
- Can be plotted with pyFoam – a Python lib
  - Very handy with lots of functions
- Smooth convergence does always mean physically realistic solutions, so beware
- But they mean something: You can stop



HyperCFD – OpenFOAM Post-Processing

- Another way to track simulations is to add probes
- Place them in the region of interest
- Values are written to a file at the end of each iteration `postProcessing/probes/timeDir/probes.dat`
- All probes are written to the same file

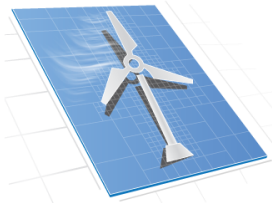
```
functions
{
    probes
    {
        // Where to load it from
        functionObjectLibs ( "libsampling.so" );

        type         probes;

        // Name of the directory for probe data
        name         probes;
        probeLocations
        (
            ( 1e-06 0 0.01 )         // at inlet
            ( 0.21 -0.20999 0.01 )   // at outlet1
            ( 0.21 0.20999 0.01 )    // at outlet2
            ( 0.21 0 0.01 )          // at central block
        );

        // Fields to be probed
        fields ( p U );

        // Write at same frequency as fields
        outputControl   outputTime;
        outputInterval  1;
    }
}
```



HyperCFD – OpenFOAM Post-Processing

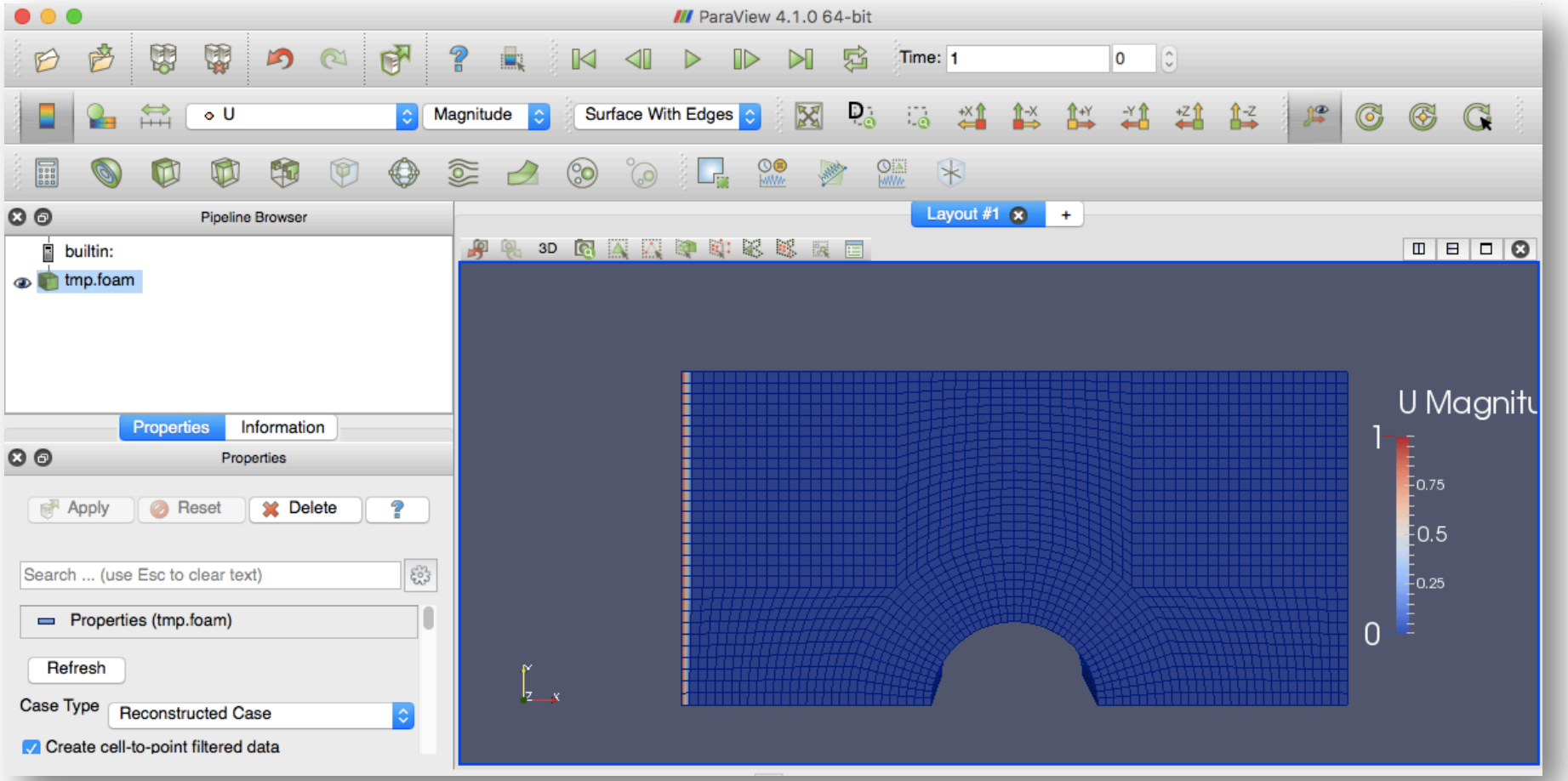
- Sampling is also a nice way of checking results. Depending on the version you're using:
  `sample`
  `postProcess -func sampleDict`
- Reads input from `system/sampleDict`
- Points, lines and surfaces can be sampled
- Suitable for checking certain regions, rotor disks, certain height above surfaces etc.

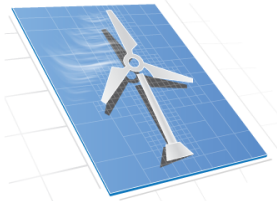
```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      sampleDict;
}
// * * * * * * * * * * * * * * * * * * * *

interpolationScheme cellPoint;

writeFormat     raw;

sampleSets
(
    uniform
    {
        name            cut;
        axis            distance;
        start           (0 0 0);
        end             (0.1 0 0);
        nPoints         100;
    }
);

fields
(
//    T
//    TPODreconstruct
    TPOD
);
```



HyperCFD – OpenFOAM Post-Processing

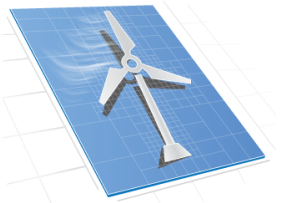
- Last but not least there is ParaView





HyperCFD – OpenFOAM Post-Processing

- It is based on VTK supports lots of filters
  - Glyphs
  - Streamlines
  - Can get screenshots
  - Can creates videos
  - Plots
  - Supports Python scripting
- You can also write shell scripts and Python scripts and call VTK functions
- This is one beauty of Linux and open source codes
- You can write a shell script that automatically creates lots of OpenFOAM cases with different inputs and then runs and saves the outputs.

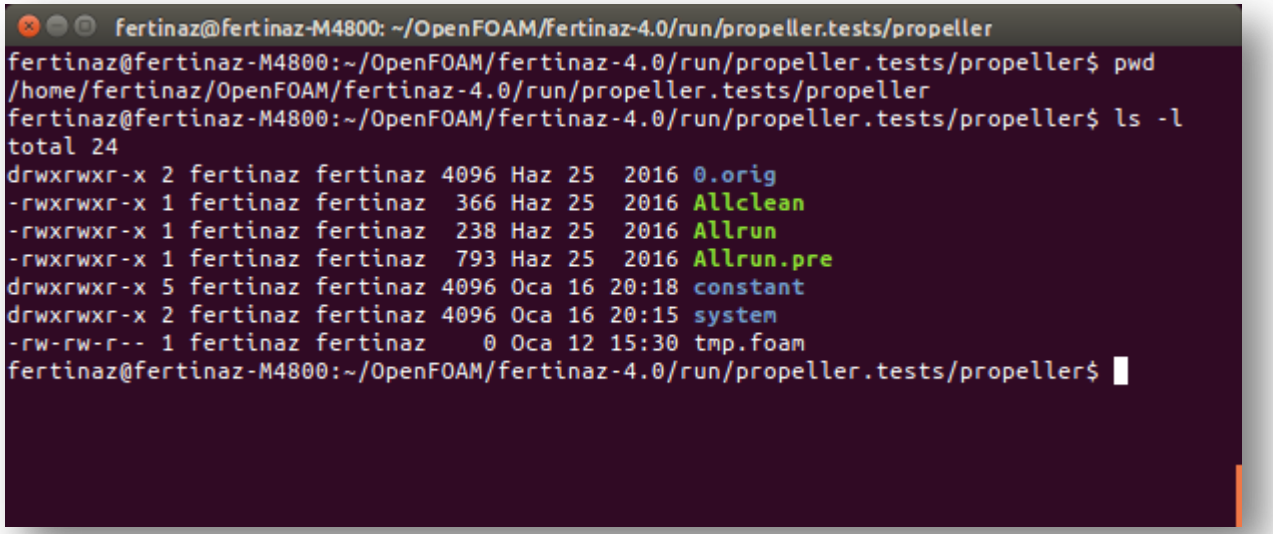


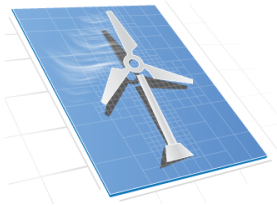
HyperCFD – OpenFOAM Sample Case: Ship Propeller

- This is not a simple case
- Copy the tutorial propeller and go to that folder

```
cp $WM_PROJECT_DIR/tutorials/incompressible/pimpleDyMFoam/propeller \
$WM_PROJECT_USER_DIR/run -r
cd $WM_PROJECT_USER_DIR/run/propeller
```

- Task – Write a shell script that automizes this procedure and runs the case.





HyperCFD – OpenFOAM Sample Case: Ship Propeller

- Take a look an the contents of the files Allrun and Allrun.pre using `cat / more / less`

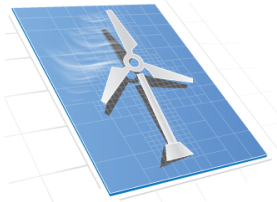
  `cat Allrun.pre`
  `cat Allrun`

fertinaz@fertinaz-M4800: ~/OpenFOAM/fertinaz-4.0/run/propeller.tests/propeller

```
fertinaz@fertinaz-M4800:~/OpenFOAM/fertinaz-4.0/run/propeller.tests/propeller$ more Allrun.pre
#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

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

# copy propeller surface from resources directory
cp $FOAM_TUTORIALS/resources/geometry/propellerTip.obj.gz constant/triSurface/


# - meshing

runApplication blockMesh

runApplication surfaceFeatureExtract

runApplication snappyHexMesh -overwrite

runApplication renumberMesh -overwrite

# force removal of fields generated by snappy
rm -rf 0

# - generate face/cell sets and zones
runApplication topoSet -dict system/createInletOutletSets.topoSetDict

# - create the inlet/outlet and AMI patches
runApplication createPatch -overwrite

# - test by running moveDynamicMes
#runApplication moveDynamicMesh -checkAMI

# - set the initial fields
cp -rf 0.orig 0
fertinaz@fertinaz-M4800:~/OpenFOAM/fertinaz-4.0/run/propeller.tests/propeller$
```

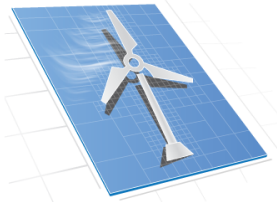
fertinaz@fertinaz-M4800: ~/OpenFOAM/fertinaz-4.0/run/propeller.tests/propeller

```
fertinaz@fertinaz-M4800:~/OpenFOAM/fertinaz-4.0/run/propeller.tests/propeller$ more Allrun
#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

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

./Allrun.pre

runApplication decomposePar

runParallel `getApplication`

runApplication reconstructPar
fertinaz@fertinaz-M4800:~/OpenFOAM/fertinaz-4.0/run/propeller.tests/propeller$
```



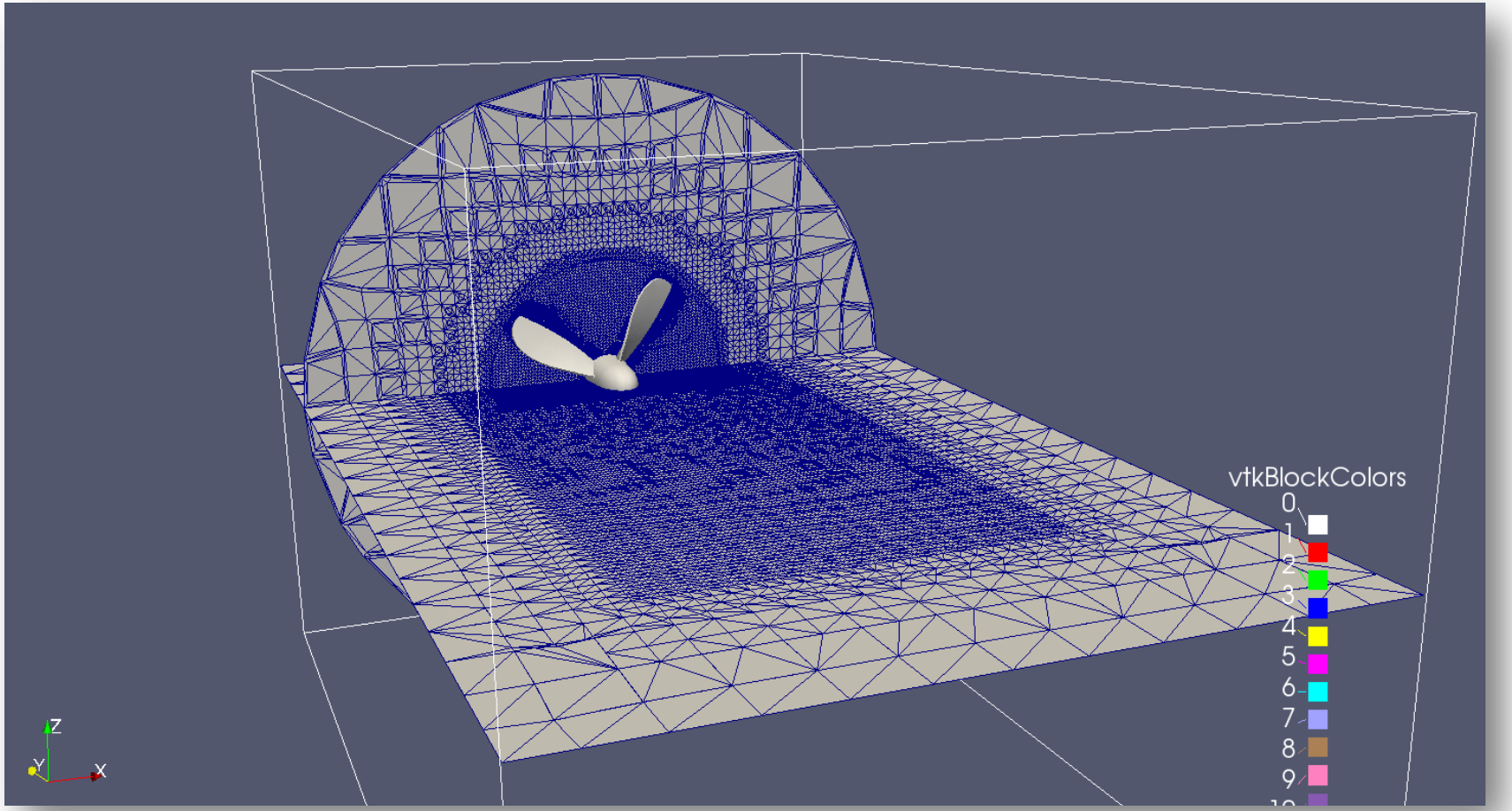
HyperCFD – OpenFOAM Sample Case: Ship Propeller

- Once we call `Allrun`, it first invokes `Allrun.pre` for preprocessing the case.
- Then case is decomposed and parallel execution start.
- Lets check what goes on step by step
  - Initial grid is generated: `blockMesh`
  - Read .obj files to detect edges of the object: `surfaceFeatureExtract`
  - Create the final mesh: `snappyHexMesh -overwrite`
  - Apply matrix reordering to improve linear solver efficiency: `renumberMesh`
  - Generate face/cell sets and zones: `topoSet`
  - Create the inlet/outlet and AMI patches: `createPatch -overwrite`



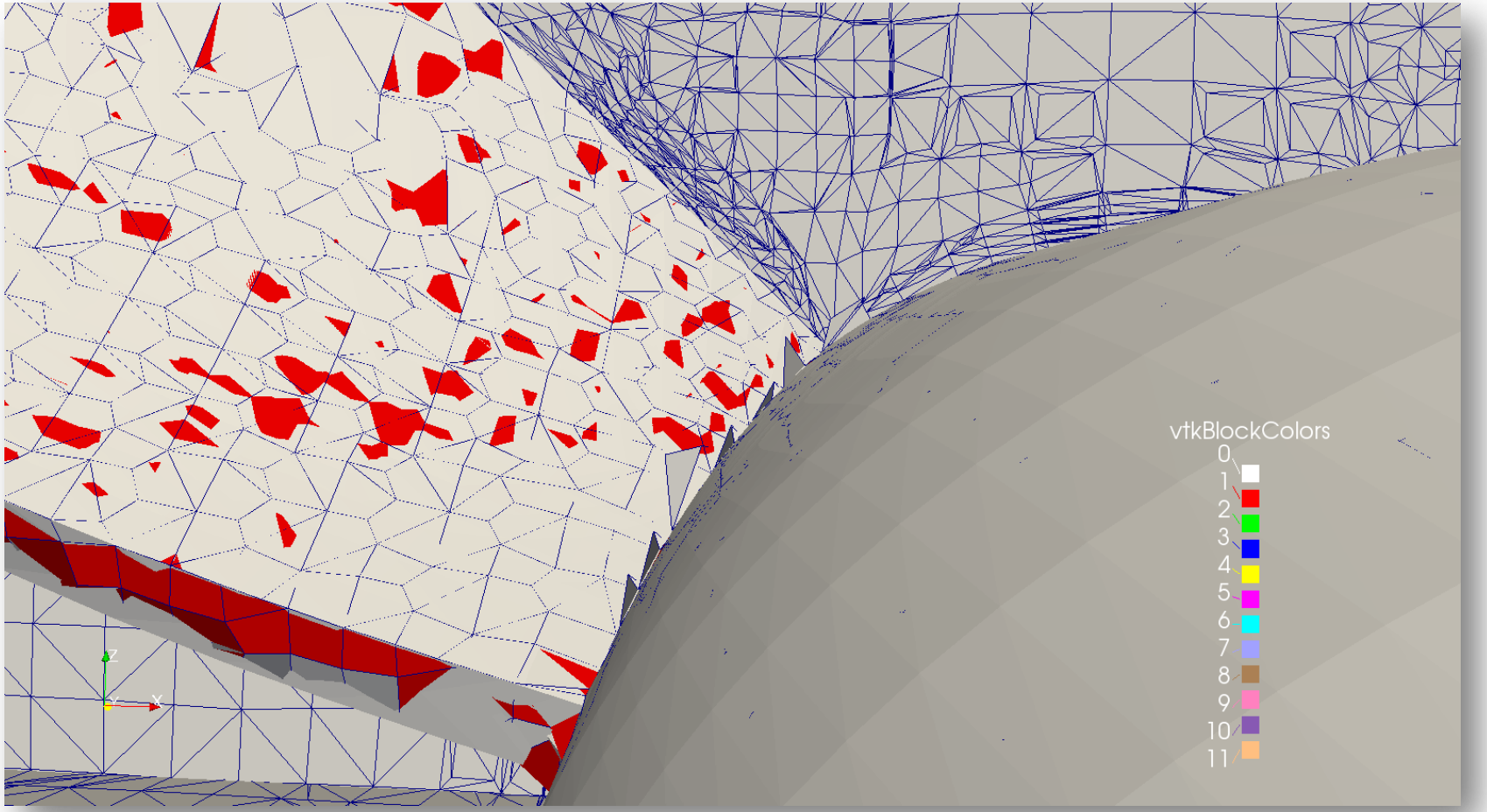
HyperCFD – OpenFOAM Sample Case: Ship Propeller

- This is the output you get after meshing – Done with ParaView using slices



HyperCFD – OpenFOAM Sample Case: Ship Propeller

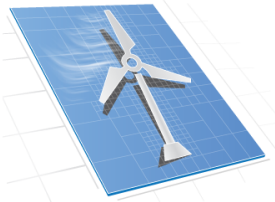
- Another view



HyperCFD – OpenFOAM Sample Case: Ship Propeller

- This is the output of `checkMesh`
- As you can see mesh seems fine

```
Checking geometry...
    Overall domain bounding box (-0.3 -0.8 -0.301742) (0.3 0.2 0.301742)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (-1.30434e-17 -8.24192e-17 -6.8355e-18) OK.
    Max cell openness = 3.5353e-16 OK.
    Max aspect ratio = 9.48442 OK.
    Minimum face area = 3.7504e-09. Maximum face area = 0.00387404.  Face area magnitudes OK.
    Min volume = 2.93457e-11. Max volume = 0.000171291.  Total volume = 0.279203.  Cell volumes OK.
    Mesh non-orthogonality Max: 63.0491 average: 8.50742
    Non-orthogonality check OK.
    Face pyramids OK.
 ***Max skewness = 4.32438, 4 highly skew faces detected which may impair the quality of the results
  <<Writing 4 skew faces to set skewFaces
    Coupled point location match (average 0) OK.

Failed 1 mesh checks.

End
```



HyperCFD – OpenFOAM Sample Case: Ship Propeller

- Now lets take a look at physical quantities: `constant/`
- kinematic viscosity: `constant/transportProperties`

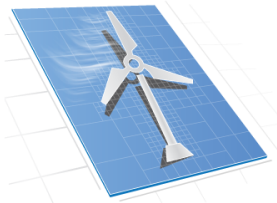
```
nu              [0 2 -1 0 0 0 0] 1e-6;
```

- turbulence: `constant/turbulenceProperties`

```
RASModel        kEpsilon;
```

- Mesh motion: `constant/dynamicMeshDict`

```
solidBodyMotionFvMeshCoeffs
{
    cellZone        innerCylinderSmall;

    solidBodyMotionFunction  rotatingMotion;
    rotatingMotionCoeffs
    {
        origin      (0 0 0);
        axis        (0 1 0);
        omega       158; // rad/s
    }
}
```

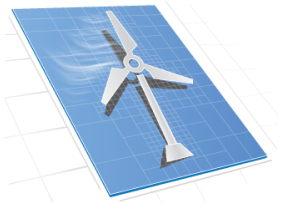


HyperCFD – OpenFOAM Sample Case: Ship Propeller

- Now lets see the boundary conditions

Boundary Type	Velocity	Pressure	TKE	Dissipation
Inlet	fixedValue	zeroGradient	fixedValue	fixedValue
Outlet	inletOutlet	fixedValue	inletOutlet	inletOutlet
Propeller.*	movingWallVelocity	zeroGradient	kqRWallFunction	epsilonWallFunction

- Inlet velocity is defined
- Pressure is `zeroGrad`
- Backflow at the outlet is handled
- Wallfunctions are used for turbulent quantities



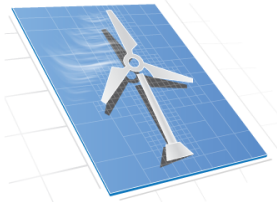
HyperCFD – OpenFOAM Sample Case: Ship Propeller

- As a next we can take a look at the discretization schemes

ddtSchemes	laplacianSchemes	interpolation
Euler	Gauss linear limited corrected 0.33	linear

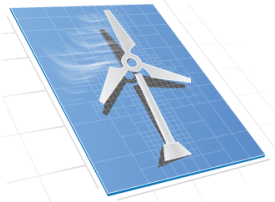
Term	gradSchemes
grad(p)	gaussLinear
grad(U)	cellLimited Gauss linear 1;

Term	divSchemes
div(phi,U)	Gauss linearUpwind grad(U)
div(phi,k)	Gauss upwind
div(phi,epsilon)	Gauss upwind
div((nuEff*dev2(T(grad(U)))))	Gauss linear



HyperCFD – OpenFOAM Sample Case: Ship Propeller

- Some comments on the discretization schemes
- Time derivation is a first order method – You can check second order method later
- For convection tutorial uses first order methods for turbulence quantities
- For gradient cell-to-cell limiter is selected and 1 means the limiter is on
- A second order accurate, bounded scheme seems to be selected for laplacian term
- So in general discretization methods seems bounded and second order accurate



HyperCFD – OpenFOAM Sample Case: Ship Propeller

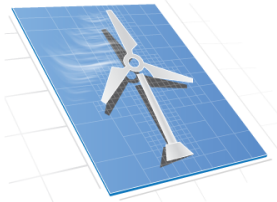
- Now we can check the linear solvers

```
pcorr
{
    solver          GAMG;
    tolerance       1e-2;
    relTol          0;
    smoother        DICGaussSeidel;
    cacheAgglomeration no;
    maxIter         50;
}

p
{
    $pcorr;
    tolerance       1e-5;
    relTol          0.01;
}

pFinal
{
    $p;
    tolerance       1e-6;
    relTol          0;
}
```

```
"(U|k|epsilon)"
{
    solver          smoothSolver;
    smoother        symGaussSeidel;
    tolerance       1e-6;
    relTol          0.1;
}

"(U|k|epsilon)Final"
{
    solver          smoothSolver;
    smoother        symGaussSeidel;
    tolerance       1e-6;
    relTol          0;
}
```

```
PIMPLE
{
    correctPhi          no;
    nOuterCorrectors    2;
    nCorrectors         1;
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    "(U|k|epsilon).*"   1;
}

cache
{
    grad(U);
}
```

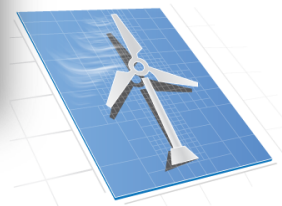
- `GAMG` is always the first option to check for pressure
- Since we use transient solver there is a final pressure corrector step
- `GaussSeidel` is also a good initial choice for asymmetric quantities



HyperCFD – OpenFOAM Sample Case: Ship Propeller

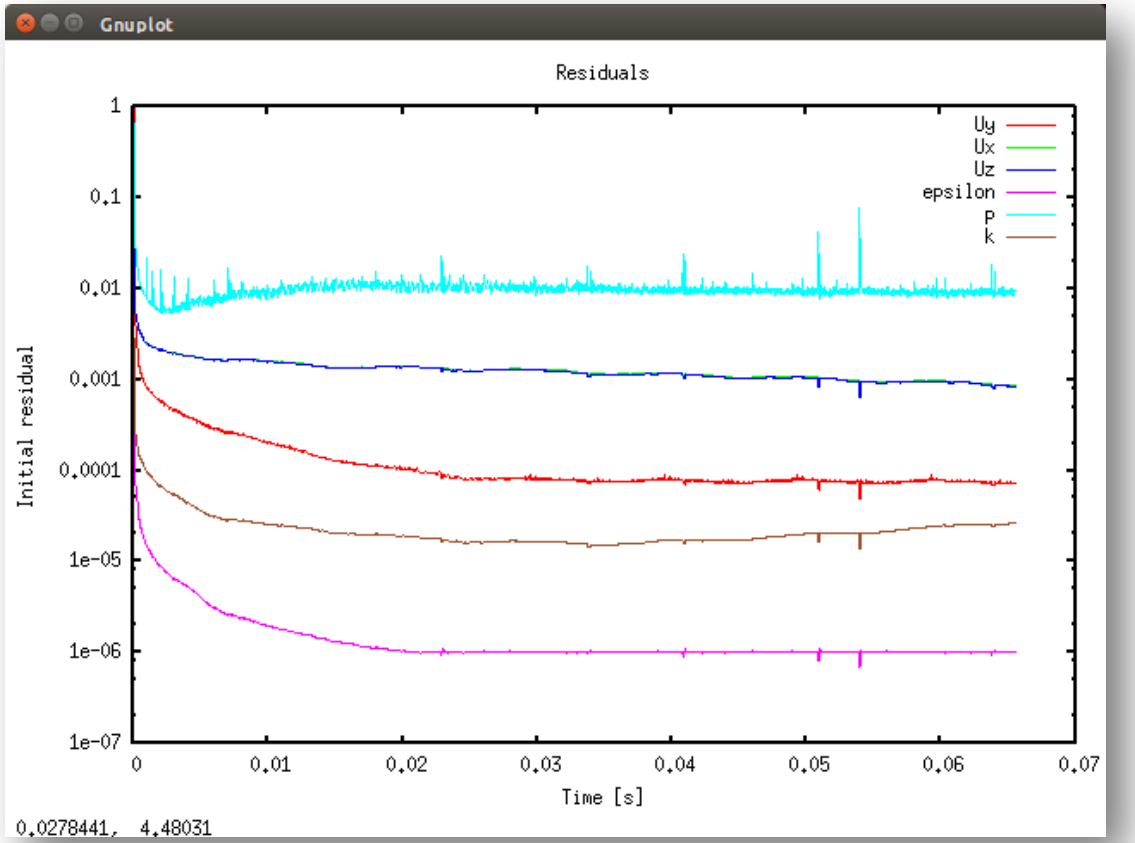
- As a final step we can check the simulation settings
  - We use the `pimpleDyMFoam` solver
  - See `maxCo`
  - This is the Courant number – see CFL condition for stability in transient solvers

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     pimpleDyMFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1;

deltaT          1e-5;
writeControl    adjustableRunTime;
writeInterval   1e-3;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  yes;

maxCo           2;

functions
{
    #includeFunc Q
    #include "surfaces"
    #include "forces"
}

// ************************************************************************* //
fertinaz@fertinaz-M4800:~/OpenFOAM/fertinaz-4.0/run/propeller.tests/propeller$
```



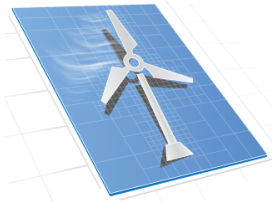
HyperCFD – OpenFOAM Sample Case: Ship Propeller

- Results can be visualized – Start with the residuals



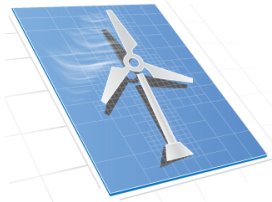
HyperCFD – OpenFOAM Sample Case: Ship Propeller

- Results can be visualized – Initial flow in y direction



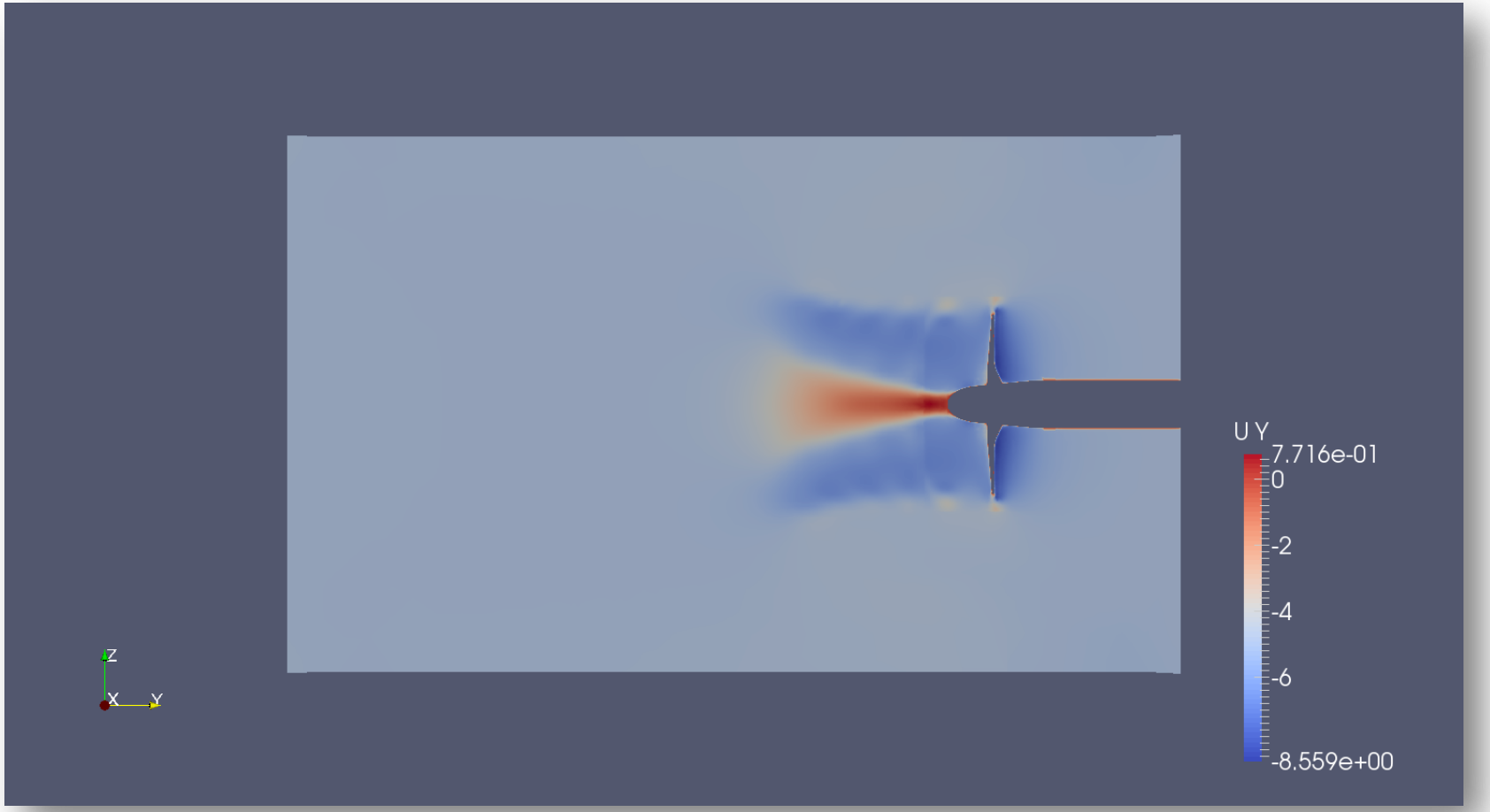
HyperCFD – OpenFOAM Sample Case: Ship Propeller

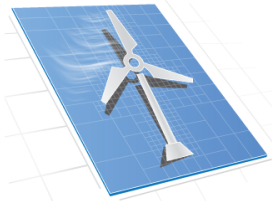
- Results can be visualized – Flow in y direction at the next time step time = 0.001



HyperCFD – OpenFOAM Sample Case: Ship Propeller

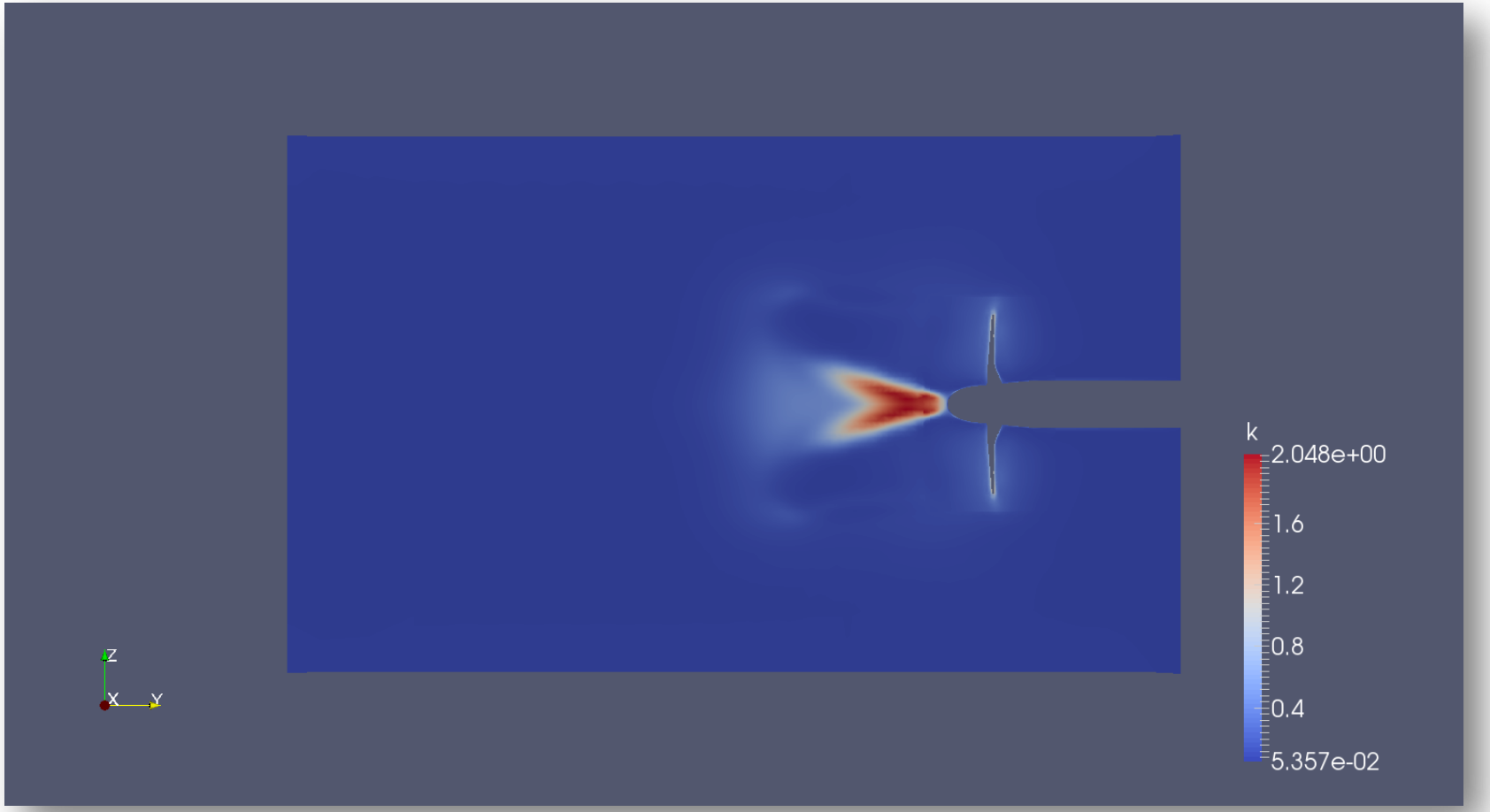
- Results can be visualized – Flow in y direction at time = 0.067

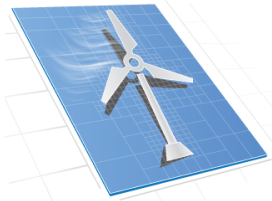




HyperCFD – OpenFOAM Sample Case: Ship Propeller

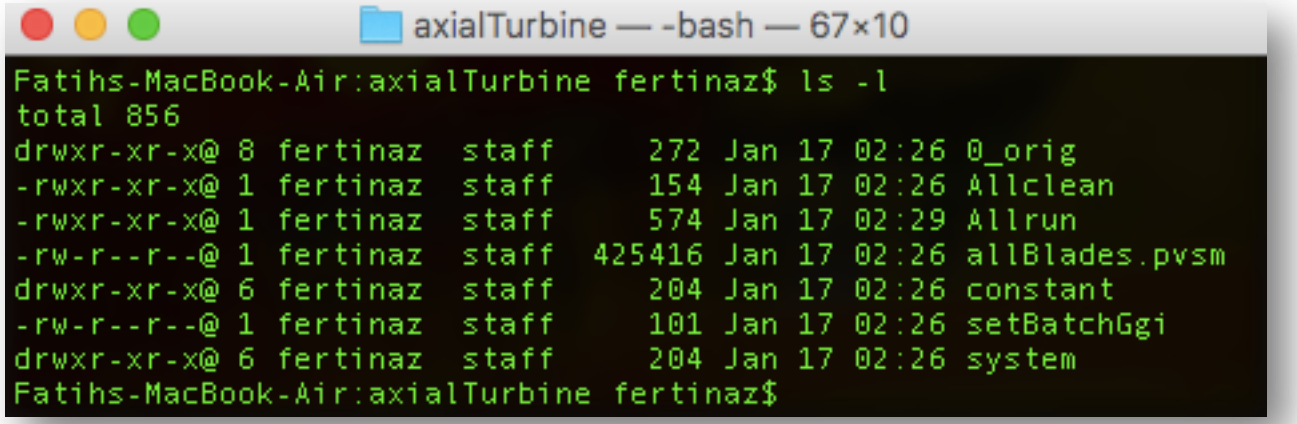
- Results can be visualized – Turbulent kinetic energy at time = 0.067

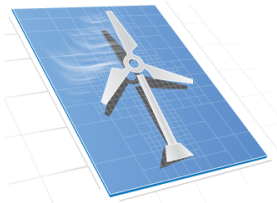




HyperCFD – OpenFOAM Sample Case: Axial Turbine

- This time I am using foam-extend 3.1 version
- Apply the same procedure – Copy axialTurbine tutorial

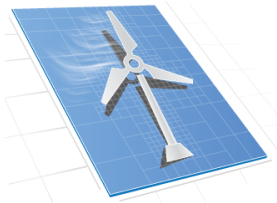




HyperCFD – OpenFOAM Sample Case: Axial Turbine

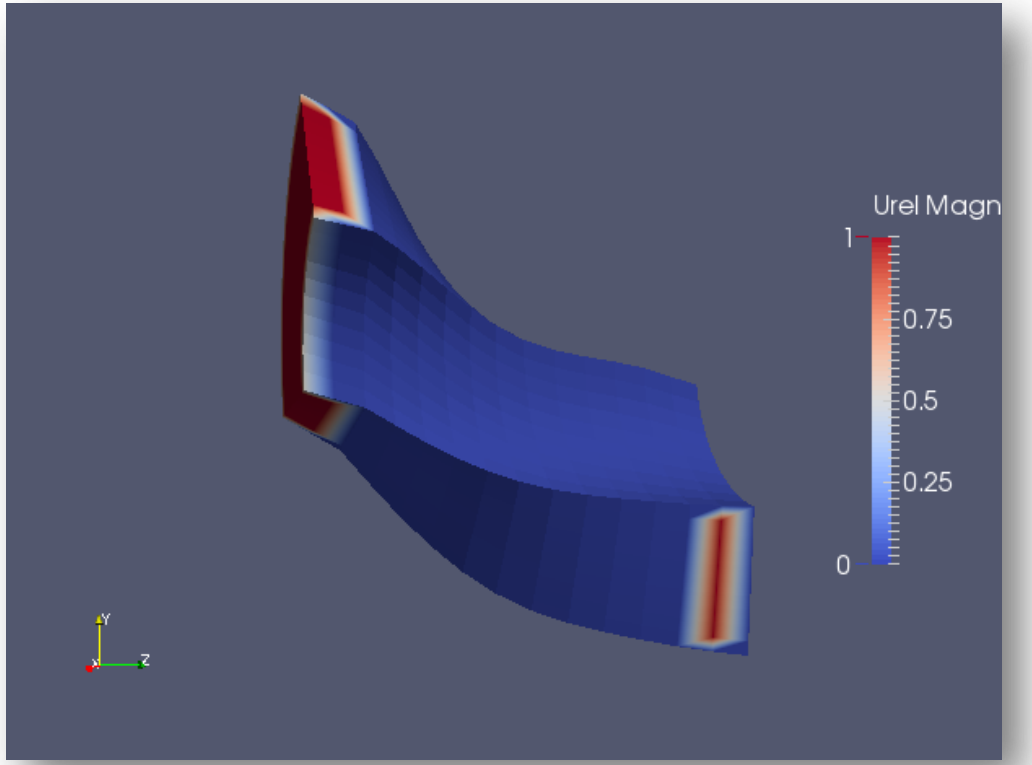
- See the computation steps

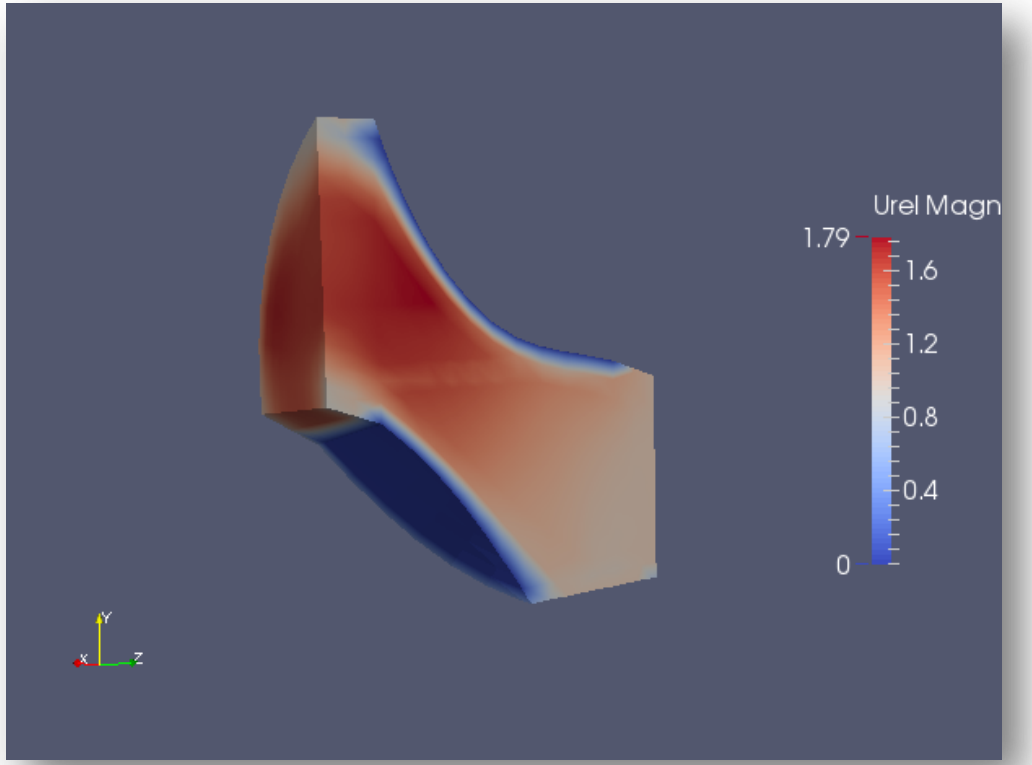
```
Fatihs-MacBook-Air:axialTurbine fertinaz$ more Allrun
#!/bin/bash
# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

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

# make sure the application is compiled
#wmake ../$application

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))"

# Set 0-directory and create GGI set:
cp -r 0_orig 0
runApplication setSet -batch setBatchGgi
runApplication setsToZones -noFlipMap

# runApplication $application
```

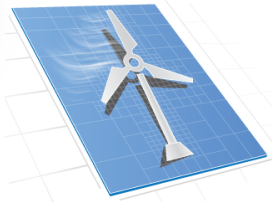


HyperCFD – OpenFOAM Sample Case: Axial Turbine

- Results – Initial state of relative velocity on the LHS and RHS is the final solution







HyperCFD – OpenFOAM Sample Case: Axial Turbine

- I would like to make a few comments at this point
- Solver is called `simpleSRFFoam:` Single rotating frame of reference
- Computes centrifugal and coriolis forces
- MRF and GGI are common techniques and available in the OpenFOAM framwork