Chalmers University of Technology

CFD with OpenSource software 2010

Implementation of an actuator disk in OpenFOAM

Developed for OpenFOAM-1.5-dev

Author:
Erik Svenning

Peer reviewed by:
Jelena Andric
Johan Magnusson

October 30, 2010

Contents

1 Introduction **2**

2 Theoretical background **2**

3 Geometrical definition of an actuator disk **4**

4 The *fan* boundary condition in OpenFOAM **5**

5 Implementation of an actuator disk model **8**
5.1 Requirements on the new model 8
5.2 General model structure 8
5.3 Modification of *simpleFoam* 8
5.4 Implementation of an actuator disk class 10
5.5 actuatorDiskExplicitForce.h 13
5.6 actuatorDiskExplicitForce.cpp 14
5.7 actuatorDiskExplicitForceSimpleFoam.C 20
5.8 UEqn.H 21
5.9 createFields.H 22

6 Basic validation **22**

7 Comparison with a case from the literature **28**

8 Summary **34**

A Proof of formulas for $A_x and \mathbf{A}_\theta$ **34**
A.1 A_x 34
A.2 A_θ 35

B Case files for the tunnel blockage case. **36**
B.1 fvSolution 36
B.2 blockMeshDict 37

1 Introduction

This tutorial shows how to implement an actuator disk in OpenFOAM. The background to the problem is described and the fan boundary condition, which is similar to the present problem, is reviewed. The implementation of an actuator disk is described in detail. Finally, the properties of the new implementation are demonstrated and investigated.

2 Theoretical background

In many cases, the flow around a propeller, fan or turbine is of interest. Often it is not possible to resolve the flow around the propeller exactly to an acceptable computational cost. An alternative to resolving the flow around the propeller exactly is to define an actuator disk region at the location of the propeller. In this way, the propeller is not modelled exactly, but the momentum transfered from the propeller to the fluid is predicted and added to the fluid within the actuator disk region. The momentum may be added as a volume source or it may be added by defining a boundary condition at the location of the actuator disk and prescribing a pressure jump.

The momentum transferred to the fluid may be predicted in different ways. The easiest approach is to add a uniform volume force over the actuator disk region and to compute this volume force in such a way that a desired total thrust and torque are achieved. A slightly more sophisticated approach is to apply a non-uniform volume force calculated from some kind of explicit equations available. An even more advanced approach would be to compute the volume force from a propeller performance code, where the propeller performance is allowed to depend on the flow variables in or close to the actuator disk region. In this way, it would be possible to achieve two way coupling between the fluid simulation and the structural simulation.

In the present implementation, the momentum transferred to the fluid is added as a volume force. The volume force varies in the radial direction and it is calculated such that a prescribed total thrust and torque is imposed on the fluid. For the volume force in the axial and tangential direction, expressions similar to those given in [4] are adopted. The volume forces are computed in such a way that the force distribution approximately follows the Goldstein optimum [1], which means that the forces have a distribution of the form:

$$\boldsymbol{f}_{bx} = A_x r^* \sqrt{1 - r^*} \tag{1}$$

$$\boldsymbol{f}_{b\theta} = A_\theta \frac{r^* \sqrt{1 - r^*}}{r^* \left(1 - r_h^{'}\right) + r_h^{'}} \tag{2}$$

$$r^* = \frac{r^{'} - r_h^{'}}{1 - r_h^{'}},\ r^{'} = \frac{r}{R_P} \tag{3}$$

$$\tag{4}$$

The constants A_x and A_θ are computed by requiring that the volume force added over the actuator disk region sums up to the total prescribed thrust T and the total prescribed torque Q. This requirement is fulfilled if A_x and A_θ are computed as (see appendix 1 for a proof):

$$A_x = \frac{105}{8} \frac{T}{\pi \Delta \left(3R_H + 4R_P\right)\left(R_P - R_H\right)} \tag{5}$$

$$A_\theta = \frac{105}{8} \frac{Q}{\pi \Delta R_P \left(R_P - R_H\right)\left(3R_P + 4R_H\right)} \tag{6}$$

$$\tag{7}$$

Figure (1) shows the normalized axial volume force versus the normalized radius. As can be seen, the above equations predict a volume force distribution that has a maximum between $r = R_H$ and $r = R_P$ while it drops to zero at $r = R_H$ and $r = R_P$. As noted in [2], a curve with the shape depicted in figure (1) is a good approximation of the Goldstein optimum distribution of circulation. Figure (2) shows the tangential volume force predicted by eqn. (2) for different values of R_H/R_P.

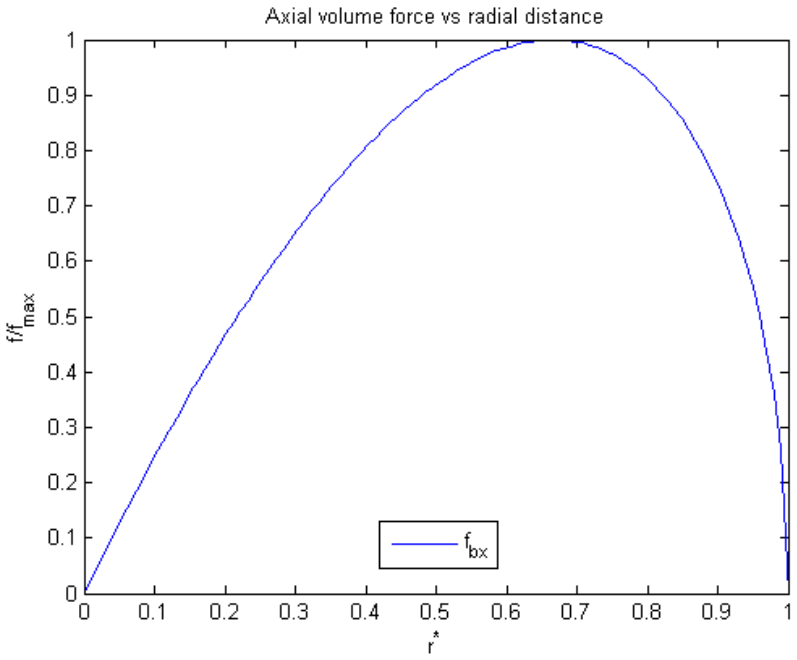

Figure 1: Normalized volume force as a function of normalized radius. The volume force is largest at moderate distances from the center line and it drops to zero for $r = R_H$ and $r = R_P$.

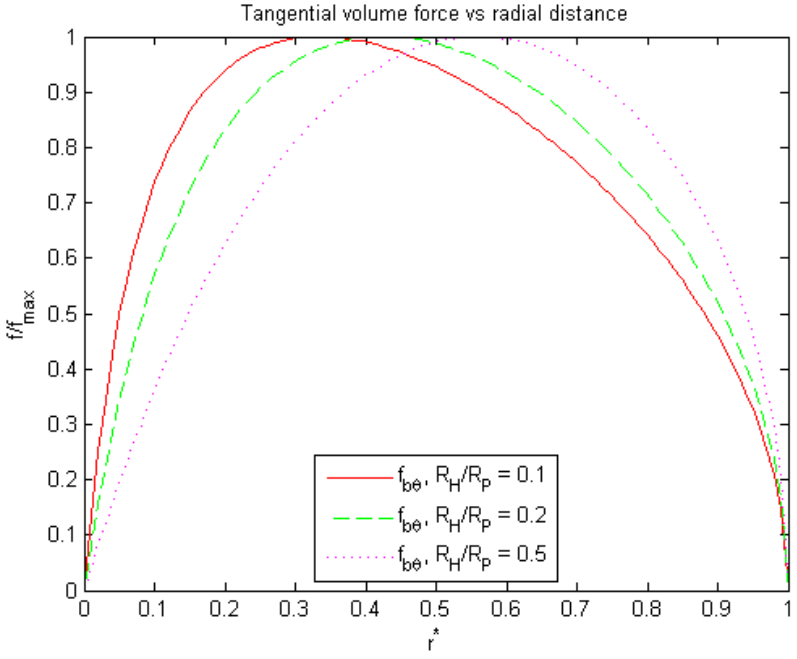

Figure 2: Volume force in the tangential direction for different values of R_H/R_P.

3 Geometrical definition of an actuator disk

In the present implementation, the actuator disk is assumed to be a hollow cylinder with interior radius R_H and exterior radius R_P. The centerline of the cylinder is defined by its start point p_s and its end point p_e. For a point to be considered to be inside the actuator disk, the following conditions must be fulfilled:

- The point must be located between the plane normal to the center line through p_s and the plane normal to the center line through p_e.
- The radial distance from the center line must be smaller than R_P.
- The radial distance from the center line must be greater than R_H.

The above conditions can be evaluated with some linear algebra:

- Compute a unit vector in the direction of the center line: $\boldsymbol{v} = \frac{\boldsymbol{p}_e - \boldsymbol{p}_s}{|\boldsymbol{p}_e - \boldsymbol{p}_s|}$
- Compute the vector from the start point to the point to be tested: $\boldsymbol{s} = \boldsymbol{q} - \boldsymbol{p}_s$
- Compute the length of the center line: $L = |\boldsymbol{p}_e - \boldsymbol{p}_s|$
- The projected length of $\boldsymbol{s}$ onto the center line is $d = \boldsymbol{s} \bullet \boldsymbol{v}$
- The point $\boldsymbol{q}$ is between the plane normal to the center line through $\boldsymbol{p}_s$ and the plane normal to the center line through $\boldsymbol{p}_e$ if $d > 0$ and $d < L$.
- Now it remains to check the radial distance. A vector from the center line to the point, that is normal to the center line, is $\boldsymbol{r} = \boldsymbol{s} - \boldsymbol{v}d$
- The radial distance is $r = |\boldsymbol{r}|$.
- The point $\boldsymbol{q}$ is located in the actuator disk region if $R_H < r < R_P$ and $\boldsymbol{q}$ is located between the two planes through $\boldsymbol{p}_s$ and $\boldsymbol{p}_e$ as described above.

The axial force $\boldsymbol{f}_{bx}$ is defined positive in the direction of $\boldsymbol{v} = \overrightarrow{\boldsymbol{p}_s \boldsymbol{p}_e}$. The tangential force $\boldsymbol{f}_{b\theta}$ has positive direction such that the moment about the center line is positive according to the right hand rule. Figure (3) shows an example of a cylindrical actuator disk region.

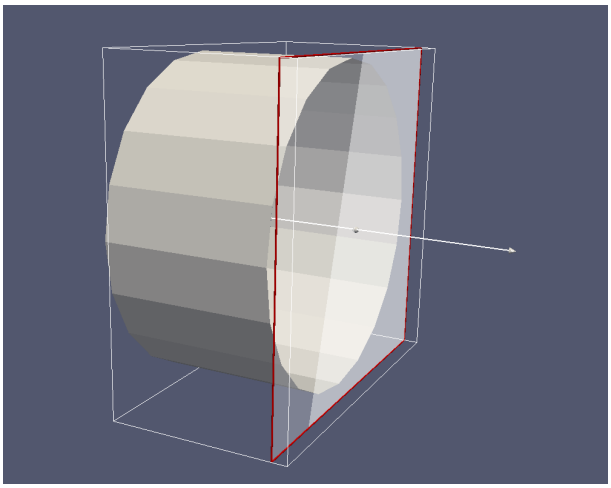

Figure 3: Outer surface of a cylindrical actuator disk.

4 The *fan* boundary condition in OpenFOAM

Before we start implementing something on our own, we will examine the *fan* boundary condition already available in OpenFOAM. Start by downloading and extracting the fan tutorial: `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/fan.tar.gz`
There is a patch called *fan*, where boundary conditions are defined. In *0/U* we find the following lines:

```
fan
{
    type            cyclic;
}
```

The turbulent kinetic energy k and the dissipation ϵ have the same boundary condition at the fan. This boundary condition just means that what comes in on one side will go out on the other side, so U, k and ϵ will be constant over the (in this example infinitely thin) actuator disk region. Now open *0/p* and locate the following lines:

```
fan
{
    type            fan;
    patchType       cyclic;
    f List<scalar>  2 (10.0 -1.0);
    value           uniform 0;
}
```

Here, the *fan* boundary condition is used, so it can be concluded that the *fan* boundary condition does something to the pressure. We will now examine what actually happens. Source OpenFOAM 1.5-dev and go to the location of the files for the *fan* bc:

```
cd $FOAM_SRC/finiteVolume/fields/fvPatchFields/derived/fan
```

In this folder, we find *fanFvPatchFields.C*, which contains the following lines:

```
void fanFvPatchField<scalar>::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    jump_ = f_[0];

    if (f_.size() > 1)
    {
        const fvsPatchField<scalar>& phip =
            patch().lookupPatchField<surfaceScalarField, scalar>("phi");

        scalarField Un =
            scalarField::subField(phip, size()/2)
           /scalarField::subField(patch().magSf(), size()/2);

        if
        (
            phip.dimensionedInternalField().dimensions()
         == dimDensity*dimVelocity*dimArea
        )
```

```
        {
            Un /= patch().lookupPatchField<volScalarField, scalar>("rho");
        }

        for(label i=1; i<f_.size(); i++)
        {
            jump_ += f_[i]*pow(Un, i);
        }
    }

    jumpCyclicFvPatchField<scalar>::updateCoeffs();
}
```

This is the part of the code that computes the pressure jump over the fan. It performs some checks and computes the velocity at the fan, then it says:

```
        for(label i=1; i<f_.size(); i++)
        {
            jump_ += f_[i]*pow(Un, i);
        }
```

We see that the pressure jump is computed as a polynomial in the velocity, where the coefficients of the polynomial, $f\_[i]$, are specified in *0/p*. Hence, the pressure jump is computed as:

$$\Delta p = \sum_{i=0}^{n} f_i \cdot U^i \tag{8}$$

where n is the number of coefficients specified in *0/p*

Now we will run the *fan* tutorial and see what happens:

```
cd fan
blockMesh
simpleFoam
paraFoam
```

Figure (4) shows the grid and the inlet, outlet and fan patches. Note that this method requires the definition of a fan patch in the mesh, the location and size of the fan can not be changed without changing the mesh.

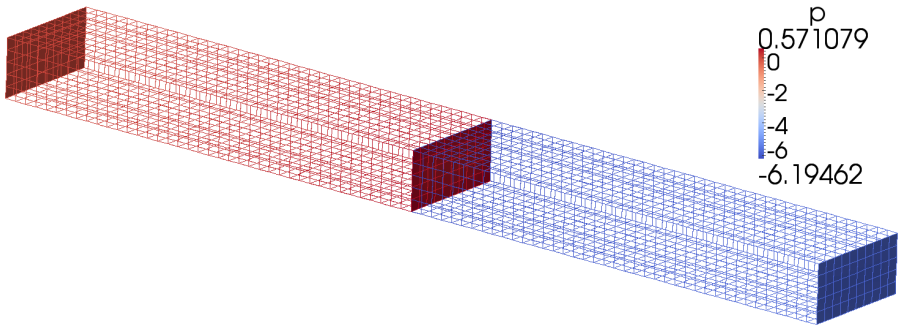

Figure 4: Geometry of the fan case. The solid patches shown are inlet (right), fan (center) and outlet (left). Note the pressure jump over the fan.

Now we will examine the pressure variation in the flow direction. Pick *Filters* → *Alphabetical* → *Plot over line* and let the line go from (0 0.5 0.25) to (8 0.5 0.25). Plot the pressure as shown in

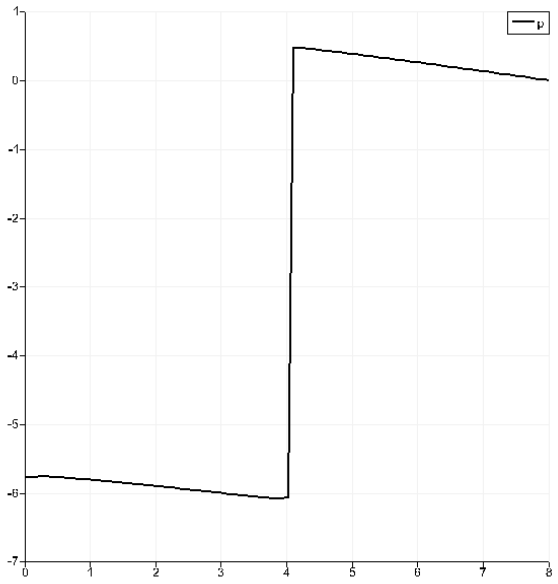

Figure 5: Pressure in the flow direction.

figure (5). We will also have a look at the streamlines to see how the *fan* bc affects the velocity field. Pick Streamline and change *Number of points* to 50 and *Line width* to 3. Keep the other settings as default. The result is shown in figure (6). The *fan* bc adds a pressure jump in the flow direction, and this is all it does. As a result, momentum is added in the axial direction, but not in the tangential direction. Hence, thrust can be modelled with this bc, but it is not possible to model swirl. Furthermore, this bc offers the posibility to add thrust as a function of the velocity through the fan, but it does not offer the possibility to add a nonuniform volume force as a function of, for example, the radial distance from the fan centerline of the point studied.

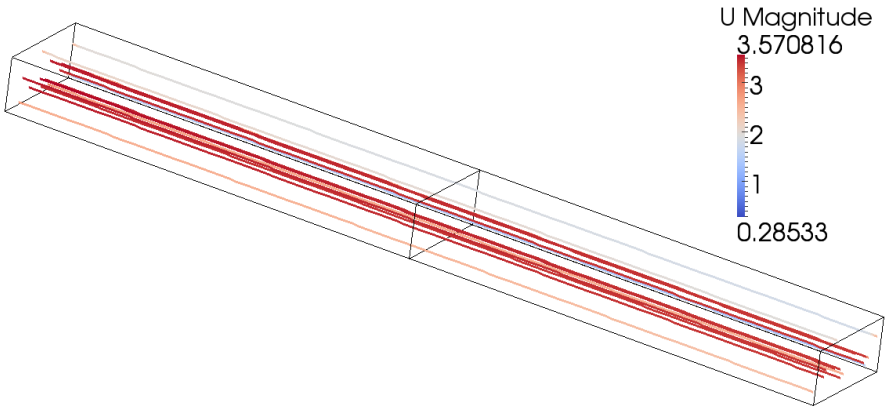

Figure 6: Streamlines of the flow through the fan. Since no swirl is added, all streamlines are straight.

5 Implementation of an actuator disk model

5.1 Requirements on the new model

The fan boundary condition is capable of modelling a pressure jump over a predefined patch. This is sufficient if only the driving force in the axial direction is of interest. However, it would be advantageuos to also have a model with the following features:

- Possibility to add a nonuniform axial force (thrust) in the actuator disk region.
- Possibility to add a nonuniform tangential force (swirl) in the actuator disk region.
- Definition of actuator disk geometry independent of mesh. It would be good to be able to add an actuator disk by just entering radius, location and some other parameters in a file, rather than having to change the mesh. To be more specific, we would like to be able to add an actuator disk without creating a patch for the actuator disk in the mesh.

5.2 General model structure

The presence of the actuator disk will be accounted for in the equations by adding a volume force in the cells corresponding to the actuator disk region. As a result of this approach, it is not possible to implement the new model as a boundary condition. Instead, the model will be implemented as a volume source and a solver will be modified to include this volume source. In principle, any solver could be chosen and the implementation procedure does not depend on the solver chosen, except that distinction must be made between compressible and incompressible solvers. Applications where the use of an actuator disk might be suitable, such as simulation of wind turbines or ships, will in general involve turbulent flow. Therefore, a solver for turbulent flow will be chosen for the application. Compressibility effects or presence of droplets or particles might be interesting, but for a start it is reasonable to chose a single phase incompressible solver to avoid introducing excessive complexity already at this early stage. Therefore, the solver *simpleFoam* is chosen for this purpose. In short, the following steps must be performed:

- Copy the simpleFoam solver and modify it to include an extra source term in the U-equation.
- Implement an actuator disk class. This class must be able to:
  - Read the geometry and other parameters relevant for an actuator disk.
  - Find all cells located in the actuator disk region and identify where in the actuator disk each cell is located (i.e. compute the radial distance from the actuator disk centerline for each cell).
  - Add a volume force to each cell in the actuator disk region according to some prescribed rule or equation. Here we will implement the equations given in the theory section, but these could easily be replaced by other equations.
  - Visualize the actuator disk in Paraview.
- Basic validation: ensure that all cells in the actuator disk region are identified properly, that the applied force field has the expected shape and that the fluid field responds as expected on a global level.
- More validation: comparison with a case from the literature.

5.3 Modification of *simpleFoam*

Copy the *simpleFoam* solver:

```
cp -r $FOAM_APP/solvers/incompressible/simpleFoam \
$WM_PROJECT_USER_DIR/applications/actuatorDiskExplicitForce
```

Rename *simpleFoam.C* to *actuatorDiskExplicitForceSimpleFoam.C* In order to be able to compile the new application, we must modify the files in the *Make* directory. Change *Make/files* to

```
actuatorDiskExplicitForceSimpleFoam.C
//actuatorDiskExplicitForce.C

EXE = $(FOAM_USER_APPBIN)/actuatorDiskExplicitForceSimpleFoam
```

Change *Make/options* to

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/turbulenceModels/RAS \
    -I$(LIB_SRC)/transportModels

EXE_LIBS = \
    -lincompressibleRASModels \
    -lincompressibleTransportModels \
    -lfiniteVolume \
    -lmeshTools \
    -llduSolvers \
    /* $(LIB_WM_OPTIONS_DIR)/libfbsdmalloc.o */
```

It is a good idea to try to compile the application before we change anything, to make sure that the basic setup is correct. Go to the *actuatorDiskExplicitForce* directory and type *wmake*. If the solver compiles without errors, we can start making our own modifications to it. Start by opening *Make/options* again and uncomment the file *actuatorDiskExplicitForce.C*, which we have not written yet:

```
actuatorDiskExplicitForceSimpleFoam.C
actuatorDiskExplicitForce.C

EXE = $(FOAM_USER_APPBIN)/actuatorDiskExplicitForceSimpleFoam
```

We want the volume force to be written to file in the same way as for example U, so that it can be visualized in Paraview. Therefore, we create a *volVectorField* to describe the volume force. Open *createFields.H* and add the following lines:

```
    Info << "Creating volume force field.\n";

    volVectorField VolumeForce
    (
        IOobject
        (
            "VolumeForce",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

To use the actuator disk, an actuator disk object must be created and initiated. This is done in *actuatorDiskExplicitForceSimpleFoam.C*. We also want to write the geometry to VTK format so that it can be visualized in Paraview. Add the following lines

```
 Info<< "\nStarting time loop\n" << endl;
```

```
    actuatorDiskExplicitForce actuatorDisk;
    //Read actuator disk geometry
    actuatorDisk.ReadGeometry(mesh);

    //Write geometry to vtk
    actuatorDisk.WriteVTK();
```

Also add the following at the top

```
#include "actuatorDiskExplicitForce.h"
```

5.4 Implementation of an actuator disk class

We will write the class *actuatorDiskExplicitForce* from scratch starting with an empty file and only write the functions we actually need, rather than copying and reusing a lot of constructors etc. This will make the code at least a little bit cleaner. The class contains the following functions and variables (a few of them are not mentioned here):

- *TypeName("actuatorDiskExplicitForce")*
- *actuatorDiskExplicitForce()* This is the default constructor and the only constructor we will implement.
- *~actuatorDiskExplicitForce();* This is the destructor.
- *void ReadGeometry(const fvMesh &iMesh)* Reads the definition of the actuator disk from *fvSolution*.
- *void CalcActuatorDiskVolForce(const fvMesh &iMesh, vectorField &ioVolumeForce)* Calculates the volume force in every point and adds it to the volume force vector field.
- *void WriteVTK()* Writes the actuator disk geometry to a vtk file.
- *vector mPointStartCenterLine*
- *vector mPointEndCenterLine*
- *scalar mExtRadius, mIntRadius*
- *scalar mThrust, mTorque*
- *scalar mRho*
- *bool PointIsInDisk(const vector &iPointStartCenterLine, const vector &iPointEndCenterLine, const vector &iPoint, scalar &oDist2, vector &oLineTangent, vector &oCircumferentialDirection)* Checks if a given point is located inside the actuator disk region.
- *scalar CalcAxialForce(const scalar iRadialDist, const scalar iRho)* Calculates the axial component of the volume force.
- *scalar CalcCircForce(const scalar iRadialDist, const scalar iRho)* Calculates the tangential component of the volume force.
- *const scalar CalcDiskThickness()* Calcualtes the thickness of the actuator disk.

There are some points in the list above that require additional explanation. The functions *CalcAxialForce()* and *CalcCircForce()* compute the scalar values of the force components in axial and tangential direction, respectively. *CalcActuatorDiskVolForce()* transforms these scalar force components to a vectorial force in cartesian coordinates and adds it to the volume force field.

Since *simpleFoam* is an incompressible solver, the momentum equation has been divided by ρ. Hence, the volume force from the actuator disk must also be divided by ρ before adding it to the momentum equation.

The VTK format is used to visualize the actuator disk region. VTK is an open source format that is capable of handling complex geometries and it is reasonably easy to implement your own visualizations in this format. See reference [5] for details. Especially, have a look at http://www.vtk.org/VTK/img/file-formats.pdf. There you can find a good description of all structures that can be visualized in VTK format. The outer surface of the actuator disk is a cylinder. It is visualized by putting points on the cylinder surface and drawing rectangular surfaces between the point. The following steps are performed in the function `WriteVTK()`:

- Initiation of variables:

```
   FILE *file;
char fileName[100];

//The cylindrical surface is visualized as 20 rectangular surfaces
unsigned int NumCells = 20;

//40 points are required; 20 points at each end of the cylinder
unsigned int NumPoints = 40;

//Number of integers needed in the the VTK file; each surface has 4 corner
//points, so we need 4 corner indices + the index of the surface = 5 indices
//per surface
unsigned int NumInts = 5*NumCells;

vectorField points(NumPoints,vector::zero);
```

- Compute a vector in the direction of the centerline of the cylinder and find a vector in the radial direction (since the cylindrical surface has rotational symmetry, any vector in the radial direction will do). Use these two vectors to form an ON-basis:

```
   vector VecLineTangent(mPointEndCenterLine - mPointStartCenterLine);
scalar LineTangentLength = sqrt(VecLineTangent.x()*VecLineTangent.x()...
...+VecLineTangent.y()*VecLineTangent.y() + VecLineTangent.z()*VecLineTangent.z());

if(LineTangentLength != 0.0) {
    VecLineTangent /= LineTangentLength;
}
else {
    Info << "Warning: The centerline tangent has zero length.\n";
    return;
}

//We need to find a vector in the radial direction. This can be any vector as long as it
// points in the radial direction. First try with (1 0 0) and see if we can
//project it onto the normal plane of the actuator disk resulting in a vector
//in the radial direction.
vector VecRadialDirection(1.0,0.0,0.0);
VecRadialDirection -= (VecRadialDirection & VecLineTangent)*VecLineTangent;

if(mag(VecRadialDirection) < SMALL) {
    //If we enter this if statement, our guess (1 0 0) was parallel to the
```

```
    //centerline of the actuator disk. Then we try (0 1 0) instead. Since
    //(1 0 0) was parallel to the centerline, (0 1 0) will for sure not be
    //parallel to the centerline. VecRadialDirection.x() = 0.0;
    VecRadialDirection.y() = 1.0;
    VecRadialDirection.z() = 0.0;

    VecRadialDirection -= (VecRadialDirection & VecLineTangent)*VecLineTangent;
}

if(mag(VecRadialDirection) > SMALL) {
    VecRadialDirection /= mag(VecRadialDirection);
}
else {
    Info << "Warning in actuatorDiskExplicitForce::WriteVTK():...
    ...mag(VecRadialDirection) close to zero.\n";
}

vector VecRadialDirection2 = VecLineTangent ^ VecRadialDirection;
scalar XLocal = 0.0, YLocal = 0.0;
```

- Compute points on the cylinder surface by starting in one point on the surface and then walking over the surface in the tangential direction:

```
   //Compute points on first side of disk region
double phi = 0.0;
for(unsigned int i = 0; i < NumCells; i++) {
    XLocal = mExtRadius*cos(phi);
    YLocal = mExtRadius*sin(phi);

    vector point(mPointStartCenterLine + XLocal*VecRadialDirection +
    YLocal*VecRadialDirection2); points[i] = point;
    phi += (1.0/double(NumCells))*2*mPI;
}

//Compute points on second side of disk region
phi = 0.0;
for(unsigned int i = 0; i < NumCells; i++) {
    XLocal = mExtRadius*cos(phi);
    YLocal = mExtRadius*sin(phi);

      vector point(mPointEndCenterLine + XLocal*VecRadialDirection+YLocal*VecRadialDirection2);
}
```

- Write everything to file:

```
        sprintf(fileName,"actuatorDisk.vtk");
file = fopen(fileName,"w");

fprintf(file,"# vtk DataFile Version 3.0\n");
fprintf(file,"Analytical surface of actuator disk. \n");
fprintf(file,"ASCII\n");
```

```
    fprintf(file,"DATASET UNSTRUCTURED_GRID\n");
    fprintf(file,"POINTS %i float\n",NumPoints);

    for(int i = 0; i < points.size(); i++) {
        fprintf(file,"%e %e %e\n",points[i].x(),points[i].y(),points[i].z());
    }

    fprintf(file,"CELLS %i %i\n",NumCells,NumInts);

    for(unsigned int i = 0; i < NumCells-1; i++) {
        fprintf(file,"%i %i %i %i %i \n",4,i,i+NumCells,i+NumCells+1,i+1);
    }
    fprintf(file,"%i %i %i %i %i \n",4,NumCells-1,2*NumCells-1,NumCells,0);

    fprintf(file,"CELL_TYPES %i\n",NumCells);

    for(unsigned int i = 0; i < NumCells; i++) {
        fprintf(file,"%i\n",9);
    }

    fclose(file);
```

The content of *actuatorDiskExplicitForce.h* and *actuatorDiskExplicitForce.cpp* is given below.

5.5 actuatorDiskExplicitForce.h

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Application
    actuatorDiskExplicitForce

Description
    Add volume force in an actuator disk region from thrust, torque and geometry defined in
        fvSolution.
    Use with actuatorDiskExplicitForceSimpleFoam

    Written by Erik Svenning, October 2010

\*---------------------------------------------------------------------------*/

#ifndef ACTUATORDISKEXPLICITFORCE_H_
#define ACTUATORDISKEXPLICITFORCE_H_

```

```
39 #include "fvCFD.H"
40
41 namespace Foam {
42
43 class actuatorDiskExplicitForce {
44
45 public:
46
47     //- Runtime type information
48     TypeName("actuatorDiskExplicitForce");
49
50
51         actuatorDiskExplicitForce();
52         ~actuatorDiskExplicitForce();
53
54         void ReadGeometry(const fvMesh &iMesh);
55         void CalcActuatorDiskVolForce(const fvMesh &iMesh, vectorField &ioVolumeForce);
56
57         void WriteVTK();
58
59 private:
60         vector mPointStartCenterLine;
61         vector mPointEndCenterLine;
62         scalar mExtRadius, mIntRadius;
63         scalar mThrust, mTorque;
64         scalar mRho;
65
66         static const double mPI = 3.141592654;
67
68         bool PointIsInDisk(const vector &iPointStartCenterLine, const vector &
              iPointEndCenterLine, const vector &iPoint, scalar &oDist2, vector &oLineTangent,
              vector &oCircumferentialDirection);
69         bool PointIsInHub(const vector &iPointStartCenterLine, const vector &
              iPointEndCenterLine, const vector &iPoint);
70
71         scalar CalcAxialForce(const scalar &iRadialDist, const scalar &iRho);
72         scalar CalcCircForce(const scalar &iRadialDist, const scalar &iRho);
73         scalar CalcDiskThickness() {return mag(mPointEndCenterLine - mPointStartCenterLine);};
74 };
75
76 }//end namespace Foam
77 #endif /* ACTUATORDISKEXPLICITFORCE_H_ */
```

5.6 actuatorDiskExplicitForce.cpp

```
1  /*---------------------------------------------------------------------------*\
2    =========                 |
3    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4     \\    /   O peration     |
5      \\  /    A nd           | Copyright held by original author
6       \\/     M anipulation  |
7  -------------------------------------------------------------------------------
8  License
9      This file is part of OpenFOAM.
10
11     OpenFOAM is free software; you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by the
13     Free Software Foundation; either version 2 of the License, or (at your
14     option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM; if not, write to the Free Software Foundation,
23     Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
24
25 Application
26     actuatorDiskExplicitForce
27
28 Description
29     Adds volume force in an actuator disk region from thrust, torque and geometry defined in
           fvSolution.
30     Use with actuatorDiskExplicitForceSimpleFoam
31
```

```
32      The actuator disk can be defined by adding the following lines in fvSolution:
33
34      actuatorDisk
35          {
36              interiorRadius       1.6; // Radius of the propeller hub
37              exteriorRadius       20.5; // Exterior radius of the propeller
38              thrust               47.5e3; // Total force in the axial direction [N]
39              torque               112.0e3; // Total torque in the actuator disk region, positive
                    according to the right hand rule
40              density              1.2; // Fluid desity
41              startPoint           (103.0 0 0); // Coordinates of start point
42              endPoint             (102.0 0 0); // Coordinates of end point
43          }
44
45      Written by Erik Svenning, October 2010
46
47
48  \*---------------------------------------------------------------------------*/
49
50  #include "actuatorDiskExplicitForce.h"
51
52  #include "faceAreaPairGAMGAgglomeration.H"
53  #include "fvMesh.H"
54  #include "surfaceFields.H"
55  #include "addToRunTimeSelectionTable.H"
56
57  namespace Foam {
58  defineTypeNameAndDebug(actuatorDiskExplicitForce, 0);
59
60
61  //Default constructor
62  actuatorDiskExplicitForce::actuatorDiskExplicitForce() {
63
64          //Set default values to all member variables
65          mPointStartCenterLine.x() = 0.0;
66          mPointStartCenterLine.y() = 0.0;
67          mPointStartCenterLine.z() = 0.0;
68
69          mPointEndCenterLine.x() = 0.0;
70          mPointEndCenterLine.y() = 0.0;
71          mPointEndCenterLine.z() = 0.0;
72
73          mExtRadius = 0.0;
74          mIntRadius = 0.0;
75          mThrust = 0.0;
76          mTorque = 0.0;
77          mRho = 1.0;
78  }
79
80  actuatorDiskExplicitForce::~actuatorDiskExplicitForce() {
81
82  }
83
84  void actuatorDiskExplicitForce::ReadGeometry(const fvMesh &iMesh) {
85          if(debug >= 1) {
86                  Info << "Reading actuator disk geometry.\n";
87          }
88
89  //
        ////////////////////////////////////////////////////////////////////////////////////////////////////
90          //Read actuator dict definition from solution dict (fvSolution)
91  //
        ////////////////////////////////////////////////////////////////////////////////////////////////////
92                  Istream& is1 = iMesh.solutionDict().subDict("actuatorDisk").lookup("
                        interiorRadius");
93              is1.format(IOstream::ASCII);
94              is1 >> mIntRadius;
95
96              Istream& is2 = iMesh.solutionDict().subDict("actuatorDisk").lookup("exteriorRadius"
                    );
97              is2.format(IOstream::ASCII);
98              is2 >> mExtRadius;
99
100             Istream& is3 = iMesh.solutionDict().subDict("actuatorDisk").lookup("thrust");
101             is3.format(IOstream::ASCII);
102             is3 >> mThrust;
103
104             Istream& is4 = iMesh.solutionDict().subDict("actuatorDisk").lookup("torque");
105             is4.format(IOstream::ASCII);
```

```
106             is4 >> mTorque;
107
108             Istream& is6 = iMesh.solutionDict().subDict("actuatorDisk").lookup("density");
109             is6.format(IOstream::ASCII);
110             is6 >> mRho;
111
112             Istream& is7 = iMesh.solutionDict().subDict("actuatorDisk").lookup("startPoint");
113             is7.format(IOstream::ASCII);
114             is7 >> mPointStartCenterLine;
115
116             Istream& is8 = iMesh.solutionDict().subDict("actuatorDisk").lookup("endPoint");
117             is8.format(IOstream::ASCII);
118             is8 >> mPointEndCenterLine;
119
120             if(debug >= 2) {
121                 Info << "Actuator disk values loaded from fvSolution:\n";
122                 Info << "mIntRadius: " << mIntRadius << "\n";
123                 Info << "mExtRadius: " << mExtRadius << "\n";
124                 Info << "mThrust: " << mThrust << "\n";
125                 Info << "mTorque: " << mTorque << "\n";
126                 Info << "mRho: " << mRho << "\n";
127                 Info << "mPointStartCenterLine: " << mPointStartCenterLine << "\n";
128                 Info << "mPointEndCenterLine: " << mPointEndCenterLine << "\n";
129             }
130
131 }
132
133 void actuatorDiskExplicitForce::CalcActuatorDiskVolForce(const fvMesh &iMesh, vectorField &
        ioVolumeForce) {
134
135         if(debug >= 1) {
136                 Info << "Calculating volume force from actuator disk.\n";
137         }
138
139         ReadGeometry(iMesh);
140
141         scalar RadialDist2;
142         vector LineTangent;
143         vector CircumferentialDirection;
144
145         vector TotalForce(0.0,0.0,0.0);
146         scalar TotalTorque = 0.0;
147
148         scalar DiskVolume = 0;
149
150         //Loop over all cells and check if the cell center is in the actuator disk region
151         for(label i = 0; i < iMesh.C().size(); i++) {
152
153                 if(PointIsInDisk(mPointStartCenterLine,mPointEndCenterLine,iMesh.C()[i],
                        RadialDist2, LineTangent,CircumferentialDirection)) {
154
155                         if(debug >= 3) {
156                                 Info << "Point: " << i << " is in the actuator disk.
                                        Coordinates: " << iMesh.C()[i] << "\n";
157                         }
158
159                         vector axialForce = LineTangent*CalcAxialForce(sqrt(RadialDist2),mRho)/
                                mRho;
160                         ioVolumeForce[i] += axialForce;
161
162                         //compute the total force added to the actuator disk, this is just for
                                control
163                         TotalForce += axialForce*iMesh.V()[i];
164
165                         vector circForce = CircumferentialDirection*CalcCircForce(sqrt(
                                RadialDist2),mRho)/mRho;
166                         ioVolumeForce[i] += circForce;
167
168                         TotalTorque += (CalcCircForce(sqrt(RadialDist2),mRho)/mRho)*sqrt(
                                RadialDist2)*iMesh.V()[i];
169                         DiskVolume += iMesh.V()[i];
170                 }
171         }
172
173         Info << "Total axial force: " << TotalForce << "\n";
174         Info << "Total torque: " << TotalTorque << "\n";
175         Info << "Total disk volume: " << DiskVolume << "\n";
176 }
177
178 void actuatorDiskExplicitForce::WriteVTK() {
```

```
179 //
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
180 //      Write the outer surface of the actuator disk to a VTK file so that it can be visualized
        in Paraview.
181 //
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
182                 FILE *file;
183                 char fileName[100];
184
185                 unsigned int NumCells = 20; //The cylindrical surface is visualized as 20
                        rectangular surfaces
186                 unsigned int NumPoints = 40; //40 points are required; 20 points at each end of
                         the cylinder
187                 unsigned int NumInts = 5*NumCells; //Number of integers needed in the the VTK
                        file; each surface has 4 corner points, so we need 4 corner indices + the
                        index of the surface = 5 indices per surface
188
189                 vectorField points(NumPoints,vector::zero);
190
191                 vector VecLineTangent(mPointEndCenterLine - mPointStartCenterLine);
192                 scalar LineTangentLength = sqrt(VecLineTangent.x()*VecLineTangent.x() +
                        VecLineTangent.y()*VecLineTangent.y() + VecLineTangent.z()*VecLineTangent.
                        z());
193
194                 if(LineTangentLength != 0.0) {
195                         VecLineTangent /= LineTangentLength;
196                 }
197                 else {
198                         Info << "Warning: The centerline tangent has zero length.\n";
199                         return;
200                 }
201
202                 //We need to find a vector in the radial direction. This can be any vector as
                        long as it points in the radial direction.
203                 //First try with (1 0 0) and see if we can project it onto the normal plane of
                        the actuator disk resulting in a vector in
204                 //the radial direction.
205                 vector VecRadialDirection(1.0,0.0,0.0);
206                 VecRadialDirection -= (VecRadialDirection & VecLineTangent)*VecLineTangent;
207
208                 if(mag(VecRadialDirection) < SMALL) {
209                         //If we enter this if statement, our guess (1 0 0) was parallel to the
                                centerline of the actuator disk. Then
210                         //we try (0 1 0) instead. Since (1 0 0) was parallel to the centerline,
                                 (0 1 0) will for sure not be parallel to
211                         //the centerline.
212                         VecRadialDirection.x() = 0.0;
213                         VecRadialDirection.y() = 1.0;
214                         VecRadialDirection.z() = 0.0;
215
216                         VecRadialDirection -= (VecRadialDirection & VecLineTangent)*
                                VecLineTangent;
217                 }
218
219                 if(mag(VecRadialDirection) > SMALL) {
220                         VecRadialDirection /= mag(VecRadialDirection);
221                 }
222                 else {
223                         Info << "Warning in actuatorDiskExplicitForce::WriteVTK(): mag(
                                VecRadialDirection) close to zero.\n";
224                 }
225
226                 vector VecRadialDirection2 = VecLineTangent ^ VecRadialDirection;
227                 scalar XLocal = 0.0, YLocal = 0.0;
228
229                 //Compute points on first side of disk region
230                 double phi = 0.0;
231                 for(unsigned int i = 0; i < NumCells; i++) {
232                         XLocal = mExtRadius*cos(phi);
233                         YLocal = mExtRadius*sin(phi);
234
235                         vector point(mPointStartCenterLine + XLocal*VecRadialDirection + YLocal
                                *VecRadialDirection2);
236                         points[i] = point;
237                         phi += (1.0/double(NumCells))*2*mPI;
238                 }
239
240                 //Compute points on second side of disk region
241                 phi = 0.0;
```

```
242                 for(unsigned int i = 0; i < NumCells; i++) {
243                         XLocal = mExtRadius*cos(phi);
244                         YLocal = mExtRadius*sin(phi);
245 
246                         vector point(mPointEndCenterLine + XLocal*VecRadialDirection + YLocal*
                                VecRadialDirection2);
247                         points[NumCells + i] = point;
248                         phi += (1.0/double(NumCells))*2*mPI;
249                 }
250 
251 
252                 sprintf(fileName,"actuatorDisk.vtk");
253                 file = fopen(fileName,"w");
254 
255                 fprintf(file,"# vtk DataFile Version 3.0\n");
256                 fprintf(file,"Analytical surface of actuator disk. \n");
257                 fprintf(file,"ASCII\n");
258 
259                 fprintf(file,"DATASET UNSTRUCTURED_GRID\n");
260                 fprintf(file,"POINTS %i float\n",NumPoints);
261 
262                 for(int i = 0; i < points.size(); i++) {
263                         fprintf(file,"%e %e %e\n",points[i].x(),points[i].y(),points[i].z());
264                 }
265 
266                 fprintf(file,"CELLS %i %i\n",NumCells,NumInts);
267 
268                 for(unsigned int i = 0; i < NumCells-1; i++) {
269                         fprintf(file,"%i %i %i %i %i \n",4,i,i+NumCells,i+NumCells+1,i+1);
270                 }
271                 fprintf(file,"%i %i %i %i %i \n",4,NumCells-1,2*NumCells-1,NumCells,0);
272 
273                 fprintf(file,"CELL_TYPES %i\n",NumCells);
274 
275                 for(unsigned int i = 0; i < NumCells; i++) {
276                         fprintf(file,"%i\n",9);
277                 }
278 
279                 fclose(file);
280 }
281 
282 
283 bool actuatorDiskExplicitForce::PointIsInDisk(const vector &iPointStartCenterLine, const vector
        &iPointEndCenterLine, const vector &iPoint, scalar &oDist2, vector &oLineTangent, vector
    &oCircumferentialDirection) {
284 //
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
285 //      Check if a given point is located in the actuator disk region.
286 //
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
287 
288         vector VecLineTangent(iPointEndCenterLine - iPointStartCenterLine);
289         scalar LineTangentLength = sqrt(VecLineTangent.x()*VecLineTangent.x() + VecLineTangent.
                y()*VecLineTangent.y() + VecLineTangent.z()*VecLineTangent.z());
290 
291         if(LineTangentLength != 0.0) {
292                 VecLineTangent /= LineTangentLength;
293         }
294         else {
295                 Info << "Warning: The centerline tangent has zero length.\n";
296                 return false;
297         }
298 
299         oLineTangent = VecLineTangent;
300 
301         vector VecStartLineToPoint(iPoint - iPointStartCenterLine);
302         scalar PointProjOnLine = VecStartLineToPoint & VecLineTangent;
303 
304         //Check if the point is inside the actuator disk in the axial direction
305         if(!(PointProjOnLine >= 0.0 && PointProjOnLine <= LineTangentLength)) {
306                 return false;
307         }
308 
309         vector VecLineToPoint(VecStartLineToPoint - (VecLineTangent*PointProjOnLine));
310         scalar RadialDist2 = VecLineToPoint.x()*VecLineToPoint.x() + VecLineToPoint.y()*
                VecLineToPoint.y() + VecLineToPoint.z()*VecLineToPoint.z();
311         oDist2 = RadialDist2;
312 
313         oCircumferentialDirection = VecLineTangent ^ VecLineToPoint;
```

```
314         oCircumferentialDirection /= mag(oCircumferentialDirection);
315
316         //Check if the point is inside the actuator disk in the radial direction
317         return(RadialDist2 <= mExtRadius*mExtRadius && RadialDist2 >= mIntRadius*mIntRadius);
318
319 }
320
321 bool actuatorDiskExplicitForce::PointIsInHub(const vector &iPointStartCenterLine, const vector
        &iPointEndCenterLine, const vector &iPoint) {
322 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
323 //      Check if a given point is located within the outer surface of the actuator disk region
        and so close to the centerline
324 //      that the radial distance is smaller than the interior radius of the actuator disk.
325 //      This function is currently not used.
326 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
327
328         vector VecLineTangent(iPointEndCenterLine - iPointStartCenterLine);
329         scalar LineTangentLength = sqrt(VecLineTangent.x()*VecLineTangent.x() + VecLineTangent.
              y()*VecLineTangent.y() + VecLineTangent.z()*VecLineTangent.z());
330
331         if(LineTangentLength != 0.0) {
332                 VecLineTangent /= LineTangentLength;
333         }
334         else {
335                 Info << "Warning: The centerline tangent has zero length.\n";
336                 return false;
337         }
338
339         vector VecStartLineToPoint(iPoint - iPointStartCenterLine);
340         scalar PointProjOnLine = VecStartLineToPoint & VecLineTangent;
341
342         //Check if the point is inside the actuator disk in the axial direction
343         if(!(PointProjOnLine >= 0.0 && PointProjOnLine <= LineTangentLength)) {
344                 return false;
345         }
346
347         vector VecLineToPoint(VecStartLineToPoint - (VecLineTangent*PointProjOnLine));
348         scalar RadialDist2 = VecLineToPoint.x()*VecLineToPoint.x() + VecLineToPoint.y()*
              VecLineToPoint.y() + VecLineToPoint.z()*VecLineToPoint.z();
349
350         //Check if the point is inside the actuator disk in the radial direction
351         return(RadialDist2 < mIntRadius*mIntRadius);
352
353 }
354
355
356 scalar actuatorDiskExplicitForce::CalcAxialForce(const scalar &iRadialDist, const scalar &iRho)
        {
357 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
358 //      Compute the force component in the axial direction. The force is computed from a simple
        equation resulting in a force
359 //      that varies with the radial distance.
360 //      If you have a better model of a rotor, comment the four lines below and add your own
        calculation of the axial force.
361 //      Do not forget to also change the calculation of the tangential force (CalcCircForce())
        below.
362 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
363         scalar axialForce = 0.0;
364         scalar radiusScaled = (iRadialDist/mExtRadius - mIntRadius/mExtRadius)/(1.0 -
              mIntRadius/mExtRadius);
365         scalar Ax = (105.0/8.0)*mThrust/(CalcDiskThickness()*mPI*(3.0*mIntRadius+4.0*mExtRadius
              )*(mExtRadius-mIntRadius));
366         axialForce = Ax*radiusScaled*sqrt(1.0 - radiusScaled);
367 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
368
369         if(debug >= 2) {
370                 Info << "Axial force: " << axialForce << "\n";
371         }
372
373         return axialForce;
374 }
```

```
375
376 scalar actuatorDiskExplicitForce::CalcCircForce(const scalar &iRadialDist, const scalar &iRho)
        {
377 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

378 //      Compute the force component in the tangential direction. The force is computed from a
        simple equation resulting in a force
379 //      that varies with the radial distance.
380 //      Change the four lines below if you have a better model.
381 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

382         scalar tangentialForce = 0.0;
383         scalar radiusScaled = (iRadialDist/mExtRadius - mIntRadius/mExtRadius)/(1.0 -
                mIntRadius/mExtRadius);
384         scalar At = (105.0/8.0)*mTorque/(CalcDiskThickness()*mPI*mExtRadius*(mExtRadius-
                mIntRadius)*(3.0*mExtRadius+4.0*mIntRadius));
385         tangentialForce = (At*radiusScaled*sqrt(1.0 - radiusScaled)/(radiusScaled*(1.0 -
                mIntRadius/mExtRadius) + mIntRadius/mExtRadius));
386 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

387
388         if(debug >= 2) {
389                 Info << "Tangential force: " << tangentialForce << "\n";
390         }
391
392         return tangentialForce;
393 }
394
395
396 } //end namespace Foam
```

5.7 actuatorDiskExplicitForceSimpleFoam.C

```
1  /*---------------------------------------------------------------------------*\
2    =========                 |
3    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4     \\    /   O peration     |
5      \\  /    A nd           | Copyright held by original author
6       \\/     M anipulation  |
7  -------------------------------------------------------------------------------
8  License
9      This file is part of OpenFOAM.
10
11     OpenFOAM is free software; you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by the
13     Free Software Foundation; either version 2 of the License, or (at your
14     option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM; if not, write to the Free Software Foundation,
23     Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
24
25 Application
26     actuatorDiskExplicitForceSimpleFoam
27
28 Description
29     Steady-state solver for incompressible, turbulent flow.
30     Modifed to allow the presence of an actuator disk.
31
32 \*---------------------------------------------------------------------------*/
33
34 #include "fvCFD.H"
35 #include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
36 #include "incompressible/RASModel/RASModel.H"
37
38 #include "actuatorDiskExplicitForce.h"
39
40 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
41
```

```
int main(int argc, char *argv[])
{

#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    actuatorDiskExplicitForce actuatorDisk;
    //Read actuator disk geometry
    actuatorDisk.ReadGeometry(mesh);

    //Write geometry to VTK
    actuatorDisk.WriteVTK();


    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "readSIMPLEControls.H"
#       include "initConvergenceCheck.H"

        p.storePrevIter();

        // Pressure-velocity SIMPLE corrector
        {
#           include "UEqn.H"
#           include "pEqn.H"
        }

        turbulence->correct();

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;

#       include "convergenceCheck.H"
    }

    Info<< "End\n" << endl;

    return(0);
}


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

5.8 UEqn.H

```
    // Solve the Momentum equation

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

    UEqn().relax();

    //Clear old values in ioVolumeForce
    for(label k = 0; k < VolumeForce.size(); k++) {
        VolumeForce[k] = vector::zero;
   }

    //Calculate volume force from actuator disk
    actuatorDisk.CalcActuatorDiskVolForce(mesh, VolumeForce);

    eqnResidual = solve
    (
```

```
        UEqn() == -fvc::grad(p) + VolumeForce
    ).initialResidual();

    maxResidual = max(eqnResidual, maxResidual);
```

5.9 createFields.H

```
    Info << "Reading field p\n" << endl;
    volScalarField p
    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

    Info << "Reading field U\n" << endl;
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

    Info << "Creating volume force field.\n";

    volVectorField VolumeForce
    (
        IOobject
        (
            "VolumeForce",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

#   include "createPhi.H"


    label pRefCell = 0;
    scalar pRefValue = 0.0;
    setRefCell(p, mesh.solutionDict().subDict("SIMPLE"), pRefCell, pRefValue);


    singlePhaseTransportModel laminarTransport(U, phi);

    autoPtr<incompressible::RASModel> turbulence
    (
        incompressible::RASModel::New(U, phi, laminarTransport)
    );
```

6 Basic validation

To ensure that the new model works somewhat reasonable, we use the *fan* tutorial discussed previously and make some changes to it. Start by copying the fan tutorial and renaming it to *cavityActuatorDisk*. Change *blockmeshDict* to:

```
convertToMeters 1;

x0 0.0;
x1 5.0;
//x2 5.0;
x3 10.0;

y0 0.0;
y1 1.0;

z0 0.0;
z1 1.0;

vertices
(
    ( $x0 $y0 $z0 )
    ( $x1 $y0 $z0 )
    ( $x1 $y1 $z0 )
    ( $x0 $y1 $z0 )
    ( $x0 $y0 $z1 )
    ( $x1 $y0 $z1 )
    ( $x1 $y1 $z1 )
    ( $x0 $y1 $z1 )
    ( $x3 $y0 $z0 )
    ( $x3 $y1 $z0 )
    ( $x3 $y0 $z1 )
    ( $x3 $y1 $z1 )
);

blocks
(
    hex (  0  1  2  3  4  5  6  7 )  ( 160 40 40 ) simpleGrading ( 1 1 1 )
    hex (  1 8 9 2 5 10 11 6 )  ( 160 40 40 ) simpleGrading ( 1 1 1 )
);

patches
(
    patch inlet
    (
        ( 0 4 7 3 )
    )
    patch outlet
    (
        ( 9 11 10 8 )
    )
    wall walls
    (
        ( 0 1 2 3 )
        ( 1 8 9 2 )
        ( 0 1 5 4 )
        ( 1 8 10 5 )
        ( 4 5 6 7 )
        ( 5 10 11 6 )
        ( 3 7 6 2 )
```

```
        ( 2 6 11 9 )
    )

//    cyclic fan
//    (
//         ( 1 2 6 5 )
//         ( 8 12 15 11 )
//    )
);

mergePatchPairs
(
);
```

As can be seen, the *fan* patch has been removed so that there is no longer an internal patch at the location of the fan. Note that some vertices have been changed, resulting in changes in the patches and blocks. We also need to remove the *fan* boundary condition in *0/U*, *0/p*, *0/k* and *0/epsilon*. When this is done, also change the internal field of U to `internalField` uniform (0 0 0); .

The new class writes the volume force to file every time step so that it can be visualized in Paraview. Hence we need a (dummy) file containing the volume force at time 0. The values specified in this file will not affect the solution, it is just a dummy file that needs to be present. For this purpose, copy the U file:

```
cp 0/U 0/VolumeForce
```

Change the dimensions in

```
0/VolumeForce
```

to

```
[0 1 -2 0 0 0 0]
```

Finally, it is time to add the definition of the actuator disk. Open *system/fvSolution* and add the following lines

```
actuatorDisk
{
   startPoint           (4.9 0.5 0.5);
   endPoint             (5.1 0.5 0.5);
   thrust               5;
   torque               0.5;
   density              1.2;
   interiorRadius       0.05;
   exteriorRadius       0.2;
}
```

Change the endTime in *system/controlDict* to

```
endTime         1000;
```

Now we can run the case and have a look at it in paraFoam:

```
actuatorDiskExplicitForceSimpleFoam >& log
paraFoam
```

Figure (7) shows the velocity magnitude on a plane with normal in the z-direction. The grey object in the center is the outer surface of the actuator disk. As can be seen, the actuator disk adds thrust to the fluid, resulting in a velocity field that is varying with the radial distance from the actuator disk centerline. It should be noted that the actuator disk only adds momentum to the fluid, it does not prevent flow through the outer surface of the actuator disk.

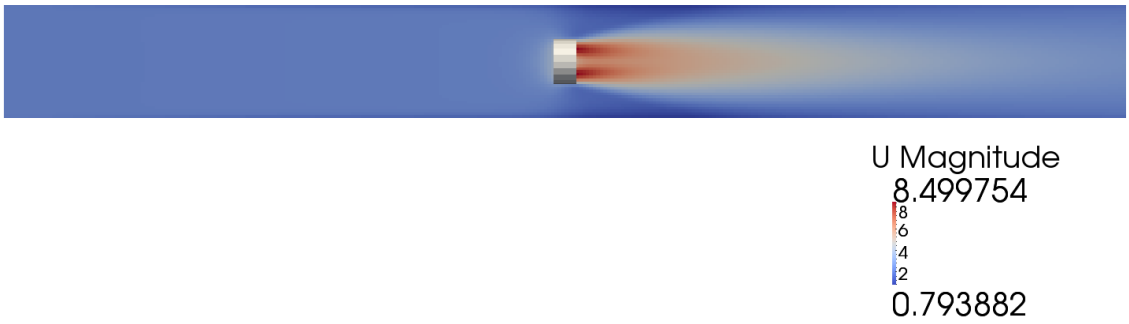

Figure 7: Velocity magnitude on a plane with normal in the y-direction

Figure (8) shows some streamlines, the outer surface of the acuator disk and arrows describing the volume force added to the fluid. The streamlines are twisted, indicating that swirl is now being imposed on the flow field. It is important that the cells belonging to the actuator disk region are identified properly and that momentum is added only to those cells. As can be seen in figures (8) and (9), all arrows describing volume force are located in the actuator disk region so no volume force is added to cells outside the actuator disk. Hence, the cells belonging to the actuator disk region have probably been identified properly. This can also be seen in figure (11), which shows the magnitude of the volume force on a slice through the domain. The white line in the figure is the outer surface of the actuator disk. As can be seen, the volume force is only nonzero in the actuator disk region. **It is once more noted that the volume force visualized in Paraview is not the "true" volume force, what we see in Paraview is f/ρ, since we are dealing with an incompressible solver.**

A note should also be made on the total volume force written to the screen during simulation. The new solver can run in parallel and will give correct results when run in parallel. However, the information written to the screen during a parallel simulation gives the impression that something is wrong, even though everything is correct. To be more specific, the wrong total force is written to the screen, but the total force added to the fluid in the simulation is correct. This reason for this is the following. When running in parallel, each processor can only "see" the cells in the part of the mesh corresponding to that processor and thus also only the part of the actuator disk corresponding to that part of the mesh. What is written to the screen is the data seen by one of the processors. An example makes this clearer: assume that we prescribe a thrust of 48 kN and that the fluid density is $1.2kg/m^3$ (see the literature case described below). Furthermore, assume that we run on 4 threads and that we split the mesh in such a way that the actuator disk region is split evenly between the 4 threads. In this case, the message written to the screen tells us that the total volume force added is only approximately $F_{tot} = 10e3$. Due to this, we could suspect that the volume force is not computed correctly. But the volume force written to the screen is only a part of the total volume force added to the fluid, to be specific it is one fourth of the total volume force since the simulation in this example was run on 4 threads and the actuator disk region was split evenly between the threads. Therefore, the total normalized volume force added is $F_{tot}/\rho = 10e3 \cdot 4 = 40e3$. Hence, the volume force added is $F_{tot} = 40e3 \cdot \rho = 48e3\ N$, so the correct volume force has indeed been added over the actuator disk region. It can be concluded that the resulting volume force is correct even though it after a quick look at the information written to the screen might seem to be wrong.

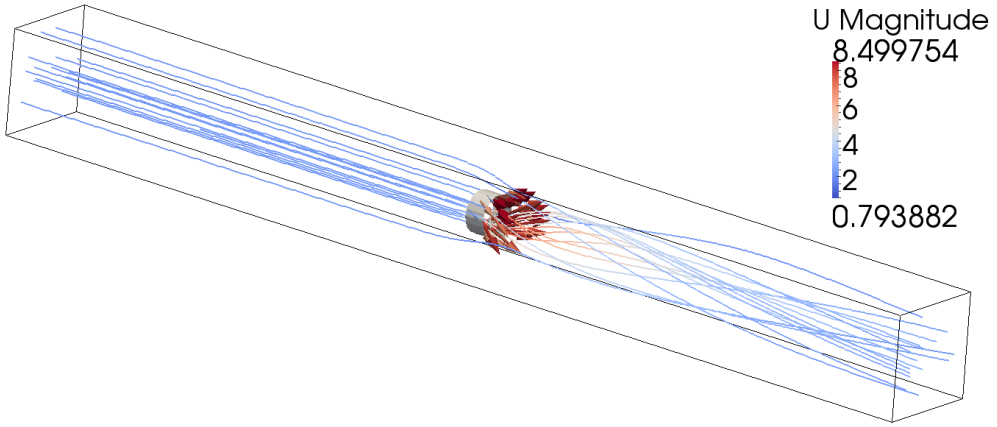

Figure 8: Streamlines colored by velocity magnitude, outer surface of actuator disk (grey object) and volume force (arrows).

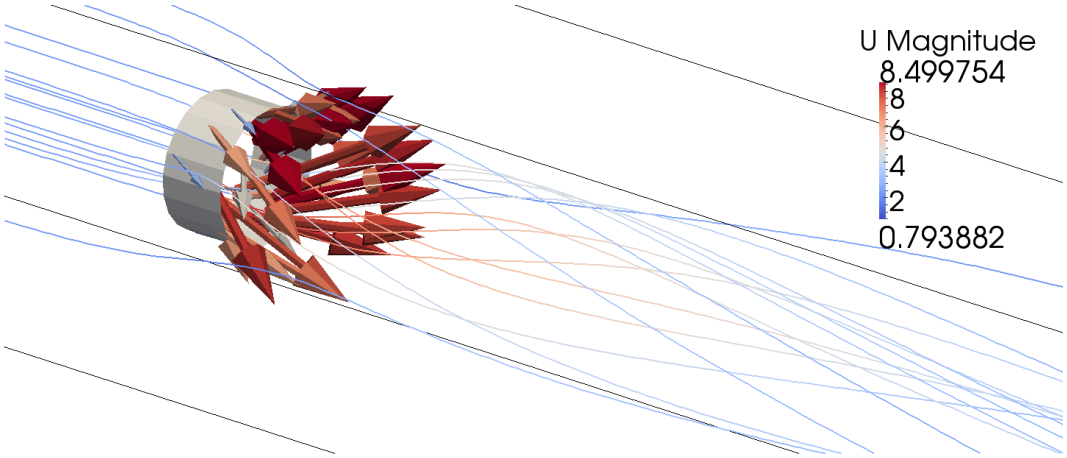

Figure 9: Magnification of the actuator disk region in the figure above.

Figure (10) shows the grid in the actuator disk region.

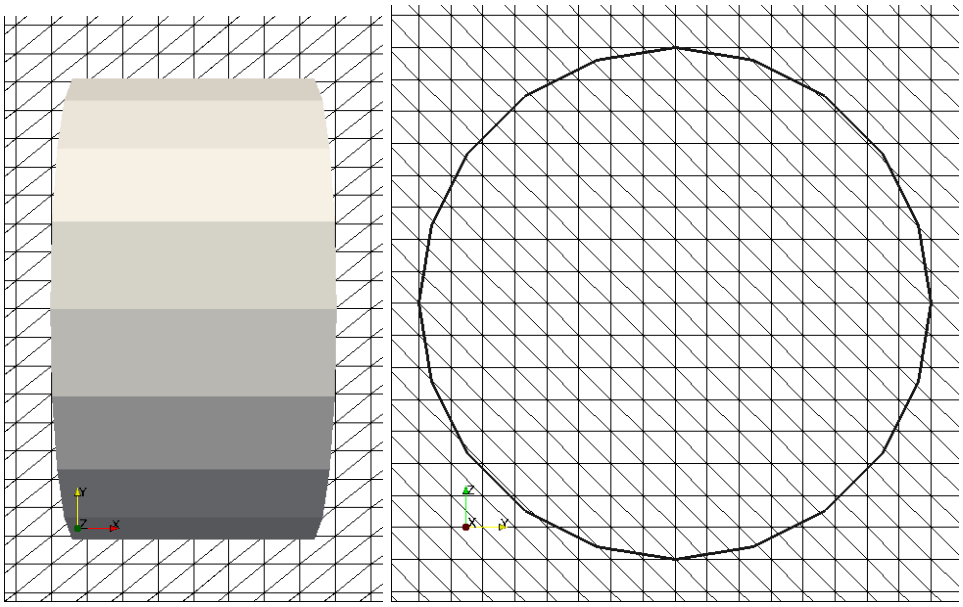

Figure 10: Grid in the actuator disk region.

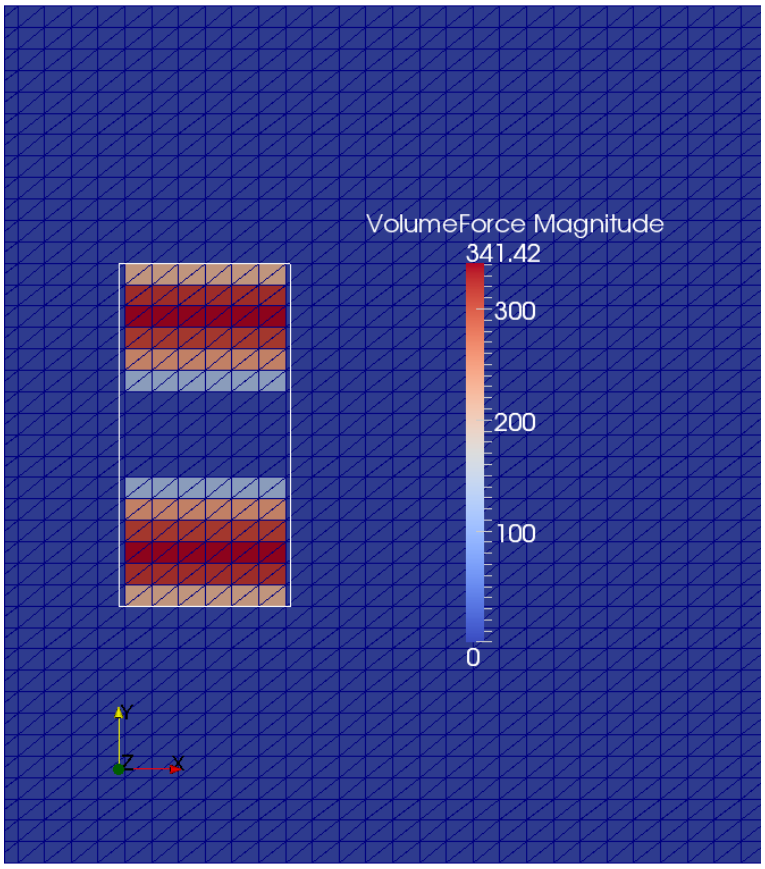

Figure 11: Volume force on a slice through the domain. The volume force is only nonzero within the actuator disk region, which is the region enclosed by the white line.

7 Comparison with a case from the literature

The current actuator disk implementation will be validated against a case described in Mikkelsen's [3] thesis. Mikkelsen studied three dimensional actuator disk models applied to wind turbines. One case considered is tunnel blockage, where a rotor affects the flow in the center of a tunnel. We will study a case with a rotor built up of LM 19.1 blades. Relevant data for the blade and the case is given in table (1).

Since the steps needed to perform a simulation with the new actuator disk class have already been described in the *Basic validation* chapter, all steps will not be described in detail here. The purpose now is to demonstrate and discuss the properties of the implementation rather than describing technicalities about running the cases.

Rotor	
Blade type	LM 19.1
Total radius	20.5 m
Thrust coefficient C_T	≈ 0.6
Power coefficient C_P	≈ 0.4
Geometry and flow data	
Freestream velocity V_0	10 m/s
Rotational speed n	27.1 rpm

Table 1: Geometry and flow data for the case considered by Mikkelsen [3].

To be able to use our own actuator disk model, we need to compute the total thrust T and the total torque Q from the above data. The estimate of the thrust and power coefficients given in table (1) have been obtained from graphs given by Mikkelsen. The thrust is computed from the thrust coefficient:

$$T = \frac{\rho V_0^2}{2} \pi R_P^2 C_T = 47.5 \ kN \tag{9}$$

The torque is computed from the power which, in turn, is computed from the power coefficient:

$$P = \frac{\rho V_0^3}{2} \pi R_P^2 C_P = 316.9 \ kW \tag{10}$$

$$n = 27.1 \Rightarrow \omega = 2.838 rad/s \tag{11}$$

$$\Rightarrow M = \frac{P}{\omega} = 112 \ kNm \tag{12}$$

$$\tag{13}$$

The size of the domain in the radial direction varies between $2R$ and $3.33R$. The size in the axial direction is chosen to be $5R$ in front of the rotor and $5R$ behind the rotor. The domain is shown in figure (12). Figure (13) shows the graded mesh in the actuator disk region. It should be noted that Mikkelsen used a cylindrical tunnel, but in the present simulations on the other hand, a rectangular tunnel is used for convenience. The boundary conditions are summarized in table (2) and the *blockMesh* and *fvSolution* files necessary for setting up the base case are given in appendix (2).

Patch	U	p	k	ϵ
Inlet	fixedValue (10 0 0)	zeroGradient	fixedValue 0.015	fixedValue 0.003
Outlet	inletOutlet	fixedValue 0	inletOutlet	inletOutlet
walls	slip	zeroGradient	zeroGradient	zeroGradient

Table 2: Summary of boundary conditions.

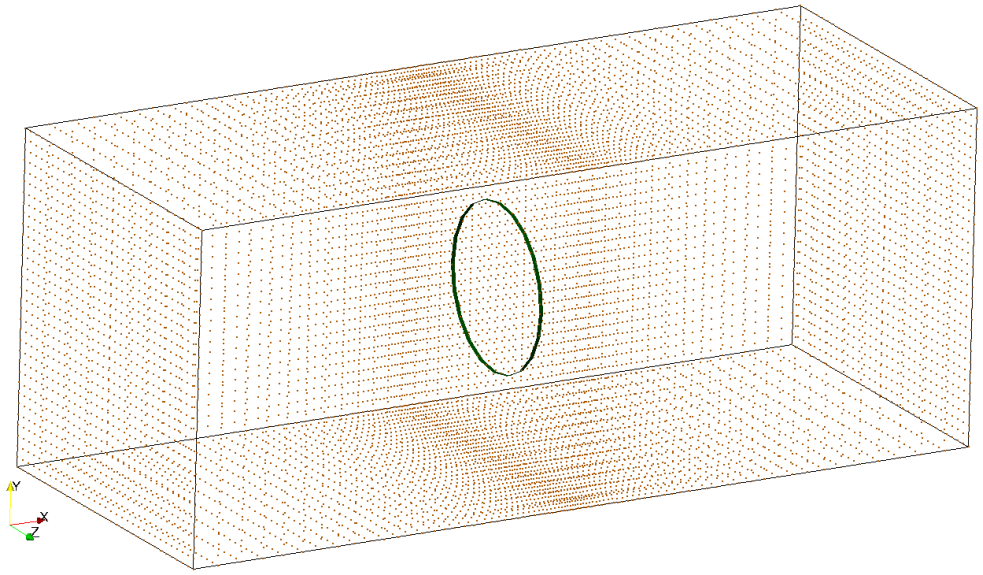

Figure 12: Domain for the tunnel blockage case. The thin ring is the actuator disk.

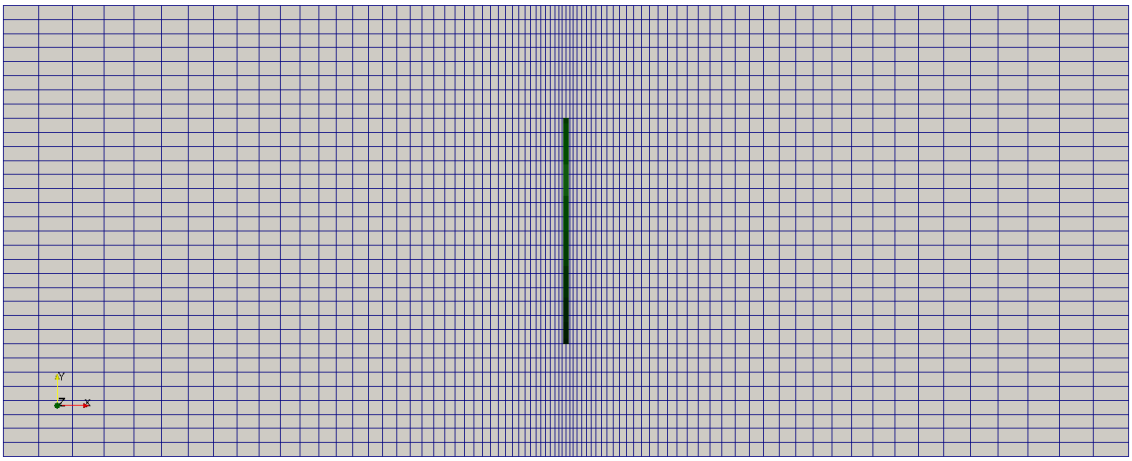

Figure 13: Grid in the actuator disk region.

Figure (14) shows the axial velocity as a function of the radial distance from the actuator disk centerline. The velocity was sampled far downstream from the actuator disk, at $x = 203$, where the velocity profile has stabilized. The curves obtained from this simulation and the results obtained by Mikkelsen [3] have the same shape except in a small region close to the actuator disk centerline ($r/R < 0.1$). The reason for this is probably that different methods where used for estimating the radial distribution of the volume force. In the present study, the volume force was assumed to obey a Goldstein optimum distribution, which results in zero volume force at $\frac{r}{R} = 0$. As a result nothing decelerates the fluid in the region close to $\frac{r}{R} = 0$. This leads to a higher velocity in this region. In the study performed by Mikkelsen, on the other hand, the radial distribution of the volume force was obtained through CFD analysis of the blade used. The result of this analysis will probably be different from the idealized Goldstein optimum distribution.

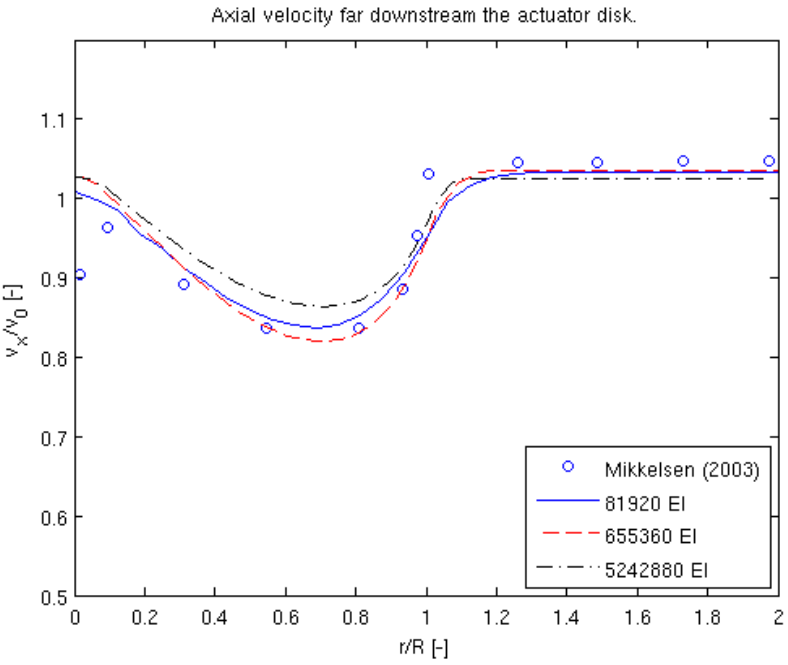

Figure 14: Grid convergence of the axial velocity profile.

Figure (15) shows the axial velocity profile predicted with different tunnel diameters. The diameter of the tunnel affects the velocity profile such that a smaller diameter shifts the velocity curve upwards. This trend is also seen in the simulations performed by Mikkelsen. Visualizations of the flow field can be seen in figures (16), (17), (18) and (19).

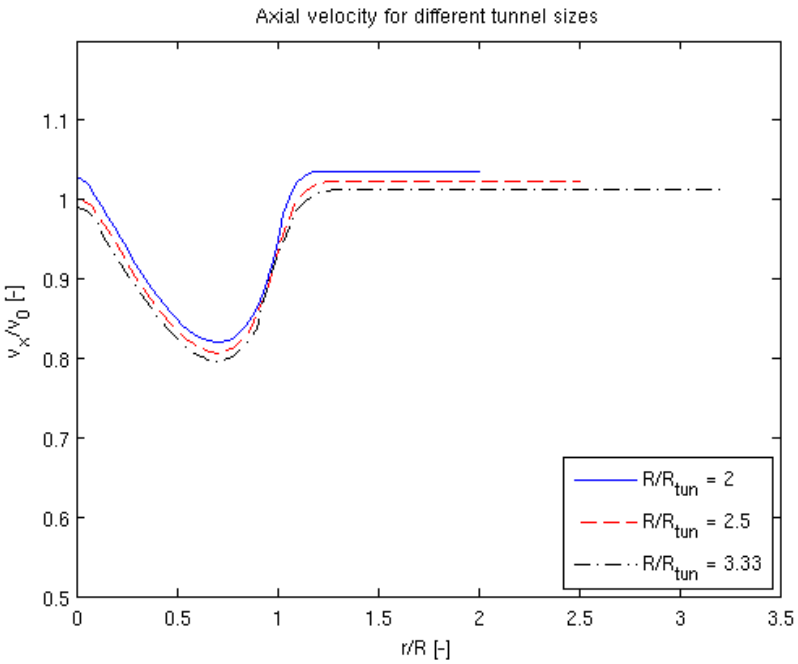

Figure 15: Axial velocity profile for different tunnel diameters.

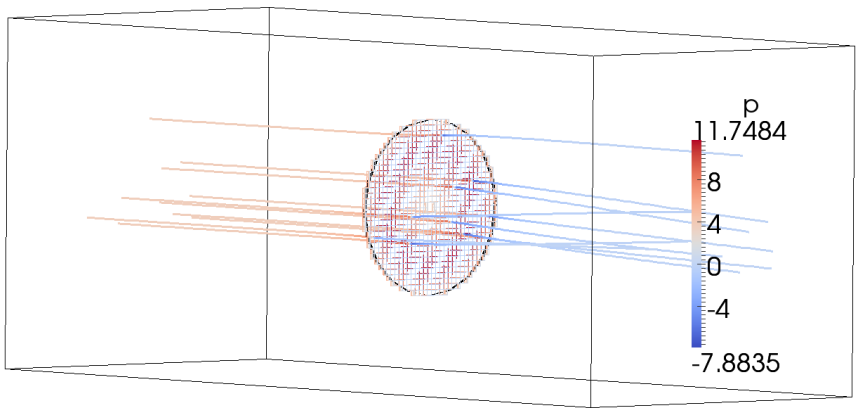

Figure 16: Streamlines and actuator disk region.

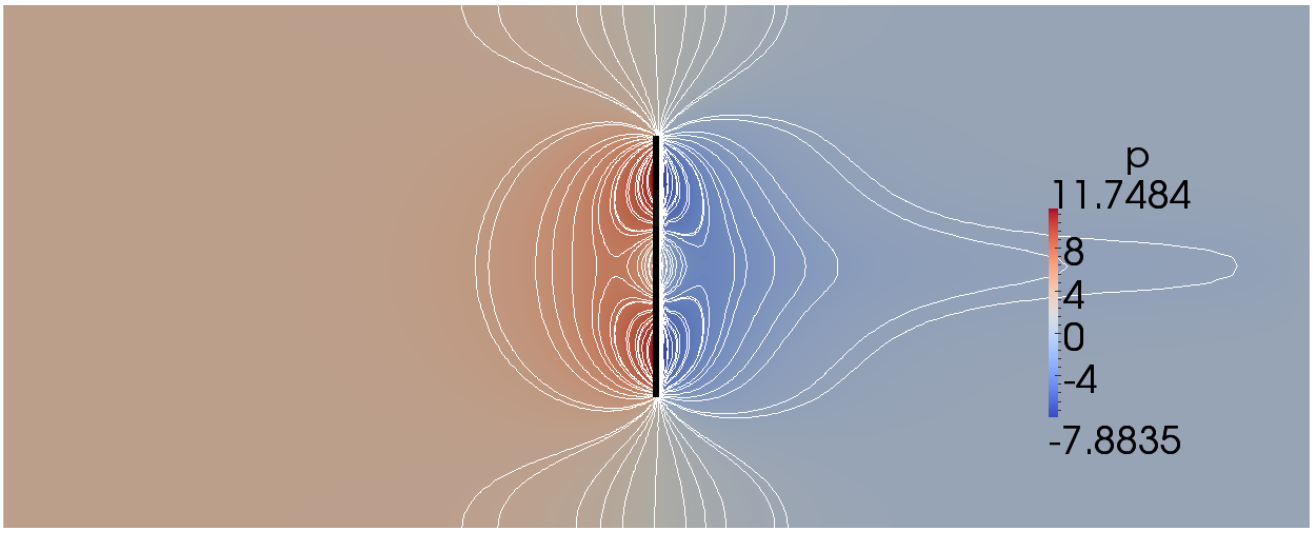

Figure 17: Pressure distribution.

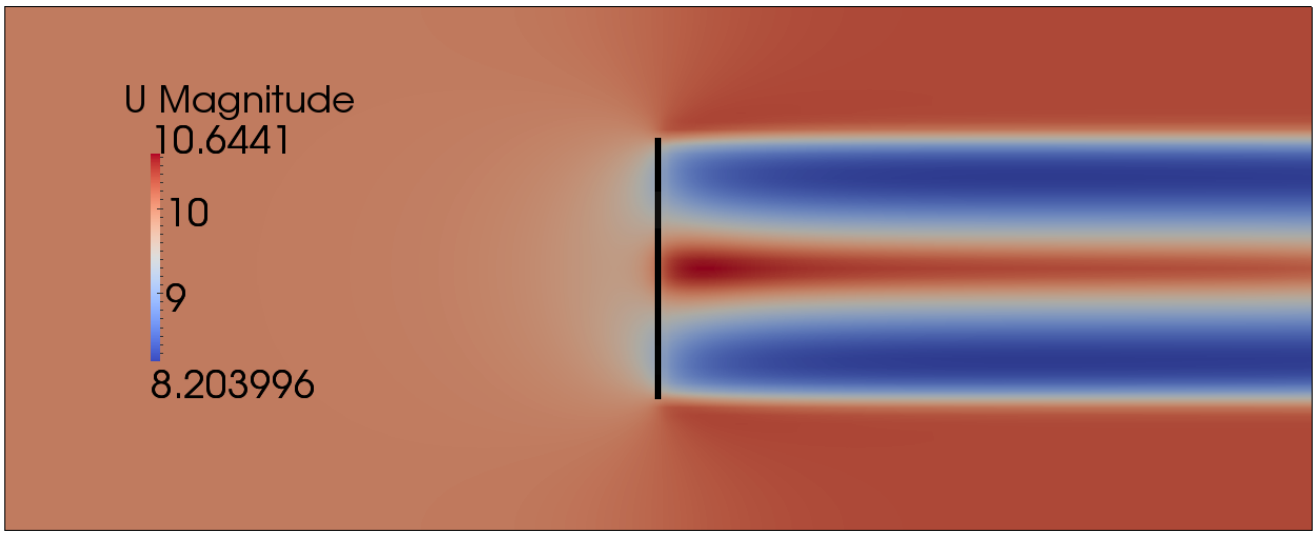

Figure 18: Velocity field.

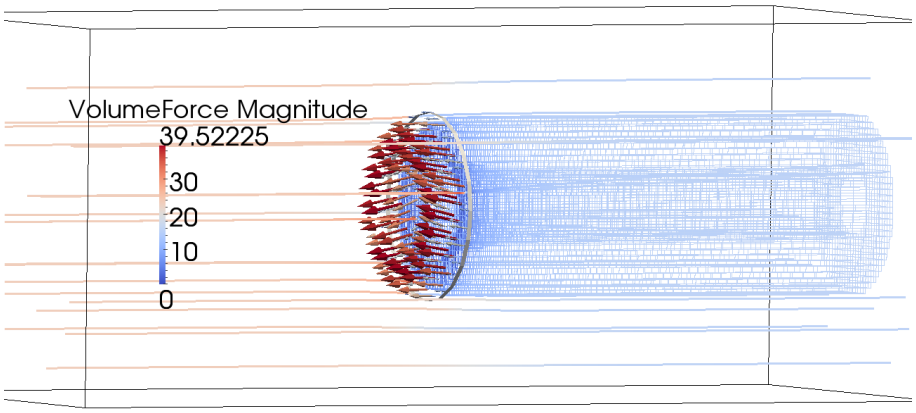

Figure 19: The arrows show the volume force added in the actuator disk region. The grey, thin cylinder is the outer surface of the actuator disk region. The blue grid is an isosurface where the velocity magnitude is 9 m/s and thus gives an indication of the extent of the region affected by the rotor. Some streamlines are also shown.

8 Summary

This tutorial has shown how to implement an actuator disk in OpenFOAM. The framework necessary for identifying the cells belonging to an actuator disk region have been identified. A volume force has been computed and added to a solver. The volume force as well as the outer surface of the actuator disk region have been visualized.

A comparison have been made with a case described in Mikkelsen's [3] thesis. The agreement is reasonable, so the comparison shows that the overall behavior of a wind turbine rotor can be predicted with the current implementation, but that discrepancies exist. It is concluded that the framework that has been established in this work can be used, but it is probably wise to consider adding a more accurate model of the volume force than simply assuming a Goldstein optimum distribution a was done in this study.

References

[1] Goldstein, S: 'On the Vortex Theory of Screw Propellers', Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character, Vol. 123, No. 792 (1929), pp. 440-465

[2] Hough, G. R. and Ordway, D. E: 'The generalized actuator disk', Technical Report TAR-TR 6401, Therm Advanced Research, Inc. (1964)

[3] Mikkelsen, R: 'Actuator Disc Methods Applied to Wind Turbines', Dissertation, Technical University of Denmark (2003), ISBN 87-7475-296-0

[4] Note on the Body Force Propeller implementation in $FINE^{TM}/Marine$, http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/Actuator_Disk.pdf

[5] The Visualization Toolkit (VTK): http://www.vtk.org

A Proof of formulas for A_x and A_θ

A.1 A_x

It is given that the axial volume force should be of the form:

$$f_{bx} = A_x r^* \sqrt{1 - r^*} \tag{14}$$

Furthermore, the volume force gives the total thrust T if:

$$T = \int_V f_{bx} dV = \int_{R_H}^{R_P} f_{bx} 2\pi r \Delta dr \tag{15}$$

Note that

$$r^* = \frac{r^{'} - r_h^{'}}{1 - r_h^{'}} \Rightarrow \frac{dr^*}{dr} = \frac{1}{R_P - R_H} \Rightarrow dr = (R_P - R_H)\, dr^* \tag{16}$$

$$r = R_H + r^* (R_P - R_H) \tag{17}$$

Insert (14) in (15) and change the variable of integration from r to r^*:

$$T = \int_{r=R_H}^{r=R_P} A_x r^* \sqrt{1-r^*} 2\pi r \Delta \left(R_P - R_H\right) dr^* = \{r = R_P \Rightarrow r^* = 1,\ r = R_H \Rightarrow r^* = 0\} = \tag{18}$$

$$= 2\pi\Delta \left(R_P - R_H\right) \int_0^1 A_x r^* \sqrt{1-r^*} \left(R_H + r^* \left(R_P - R_H\right)\right) dr^* = \tag{19}$$

$$= 2\pi\Delta \left(R_P - R_H\right) A_x \left[R_H \underbrace{\int_0^1 r^* \sqrt{1-r^*} dr^*}_{I} + \left(R_P - R_H\right) \underbrace{\int_0^1 r^{*2} \sqrt{1-r^*} dr^*}_{II} \right] \tag{20}$$

Evaluation of the integrals I and II in the expressions above gives:

$$I = \int_0^1 r^* \sqrt{1-r^*} dr^* = \frac{4}{15} \tag{21}$$

$$II = \int_0^1 r^{*2} \sqrt{1-r^*} dr^* = \frac{16}{105} \tag{22}$$

Insert the numerical values of these intergrals in (20):

$$T = 2\pi\Delta \left(R_P - R_H\right) A_x \frac{4}{105} \left(3R_H + 4R_P\right) \tag{23}$$

$$\Rightarrow A_x = \frac{105}{8} \frac{T}{\pi\Delta \left(3R_H + 4R_P\right) \left(r_P - R_H\right)} \tag{24}$$

A.2 $\mathbf{A}_\theta$

The tangential force distribution is assumed to have a distribution of the form:

$$f_{b\theta} = A_\theta \frac{r^* \sqrt{1-r^*}}{r^* \left(1 - r_h^{'}\right) + r_h^{'}} \tag{25}$$

$$r_h^{'} = \frac{R_H}{R_P} \tag{26}$$

The volume force gives the total torque Q if:

$$Q = \int_V r f_{b\theta} dV = \int_{R_H}^{R_P} r f_{b\theta} 2\pi r \Delta dr \tag{27}$$

Insert (25) in (27) change variable of integration:

$$Q = \int_{R_H}^{R_P} rA_\theta \frac{r^*\sqrt{1-r^*}}{r^*\left(1-r_h^{'}\right)+r_h^{'}} 2\pi r \Delta dr = \tag{28}$$

$$= A_\theta 2\pi\Delta \int_0^1 \left(R_H + r^*\left(R_P - R_H\right)\right)^2 \frac{r^*\sqrt{1-r^*}}{r^*\left(1-r_h^{'}\right)+r_h^{'}} \left(R_P - R_H\right) dr^* = \tag{29}$$

$$= A_\theta 2\pi\Delta \int_0^1 \left[R_P\left(r_h^{'} + r^*\left(1-r_h^{'}\right)\right)\right]^2 \frac{r^*\sqrt{1-r^*}}{r^*\left(1-r_h^{'}\right)+r_h^{'}} R_P\left(1-r_h^{'}\right) dr^* = \tag{30}$$

$$= A_\theta 2\pi\Delta R_P^3 \int_0^1 \left(r_h^{'} + r^*\left(1-r_h^{'}\right)\right) r^*\sqrt{1-r^*}\left(1-r_h^{'}\right) dr^* = \tag{31}$$

$$= A_\theta 2\pi\Delta R_P^3 \left(1-r_h^{'}\right)\left[r_h^{'}\int_0^1 r^*\sqrt{1-r^*}dr^* + \left(1-r_h^{'}\right)\int_0^1 r^{*2}\sqrt{1-r^*}dr^*\right] = \tag{32}$$

$$= A_\theta 2\pi\Delta R_P\left(R_P - R_H\right)\frac{4}{105}\left(3R_H + 4R_P\right) \tag{33}$$

$$\Rightarrow A_\theta = \frac{105}{8}\frac{Q}{\Delta\pi R_P\left(R_P - R_H\right)\left(3R_P + 4R_H\right)} \tag{34}$$

B Case files for the tunnel blockage case.

B.1 fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p PCG
    {
        preconditioner   DIC;
        tolerance        1e-07;
        relTol           0.00001;

    };
    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-07;
        relTol           0.00001;
    };
    k PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-06;
        relTol           0.00001;
    };
    epsilon PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-06;
        relTol           0.00001;
    };
    R PBiCG
    {
        preconditioner   DILU;
```

```
        tolerance        1e-05;
        relTol           0.01;
    };
    nuTilda PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0.01;
    };
}

SIMPLE
{
    nNonOrthogonalCorrectors 1;
}

actuatorDisk
{
   interiorRadius       1.6;
   exteriorRadius       20.5;
   thrust               47.5e3;
   torque               112.0e3;
   density              1.2;
   startPoint           (103.0 0 0);
   endPoint             (102.0 0 0);
}

relaxationFactors
{
    p               0.4;
    U               0.4;
    k               0.4;
    epsilon         0.4;
    R               0.4;
    nuTilda         0.4;
}

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

B.2 blockMeshDict

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.2                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
   version         2.0;
   format          ascii;

   root            "";
   case            "";
   instance        "";
   local           "";

   class           dictionary;
   object          blockMeshDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

x0 0.0;
x1 102.50;
//x2 5.0;
x3 205.0;

y0 -41.0;
y1 41.0;

z0 -41.0;
z1 41.0;
```

```
37
38  vertices
39  (
40      ( $x0 $y0 $z0 )
41      ( $x1 $y0 $z0 )
42      ( $x1 $y1 $z0 )
43      ( $x0 $y1 $z0 )
44      ( $x0 $y0 $z1 )
45      ( $x1 $y0 $z1 )
46      ( $x1 $y1 $z1 )
47      ( $x0 $y1 $z1 )
48      ( $x3 $y0 $z0 )
49      ( $x3 $y1 $z0 )
50      ( $x3 $y0 $z1 )
51      ( $x3 $y1 $z1 )
52  );
53
54  blocks
55  (
56      hex (  0  1  2  3  4  5  6  7 )  ( 80 64 64 ) simpleGrading ( 0.1 1 1 )
57      hex (  1 8 9 2 5 10 11 6 )  ( 80 64 64 ) simpleGrading ( 10 1 1 )
58  );
59
60  patches
61  (
62      patch inlet
63      (
64          ( 0 4 7 3 )
65      )
66      patch outlet
67      (
68          ( 9 11 10 8 )
69      )
70      wall walls
71      (
72          ( 0 1 2 3 )
73          ( 1 8 9 2 )
74          ( 0 1 5 4 )
75          ( 1 8 10 5 )
76          ( 4 5 6 7 )
77          ( 5 10 11 6 )
78          ( 3 7 6 2 )
79          ( 2 6 11 9 )
80      )
81
82  //    cyclic fan
83  //    (
84  //          ( 1 2 6 5 )
85  //          ( 8 12 15 11 )
86  //    )
87  );
88
89  mergePatchPairs
90  (
91  );
```