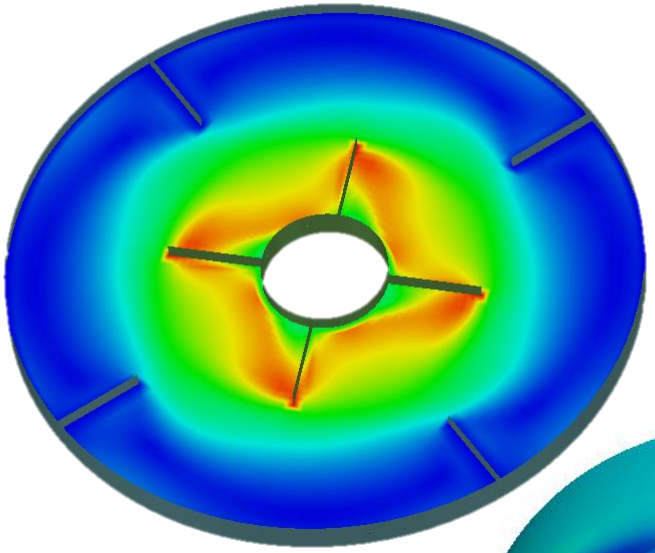
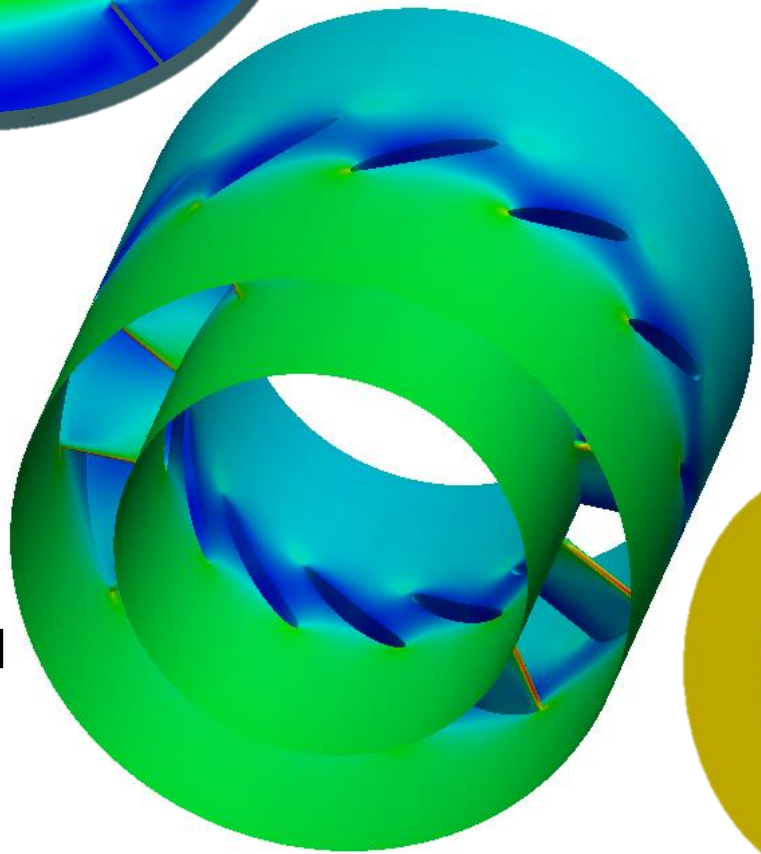
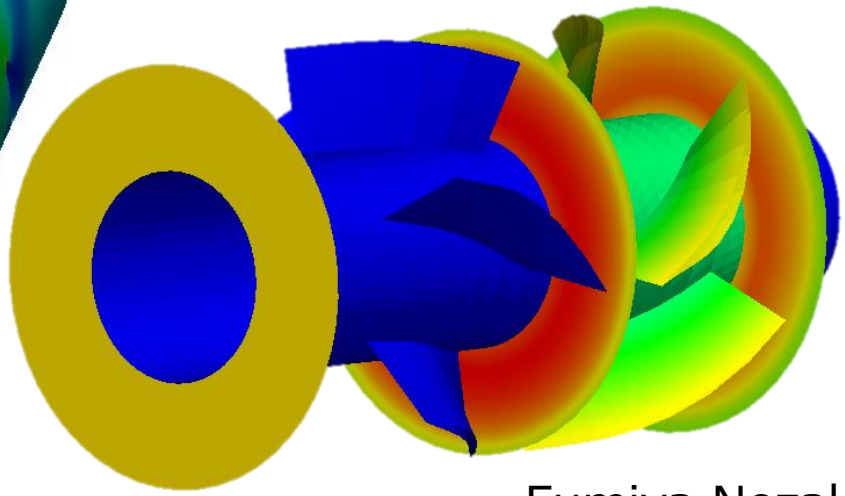
English

CFD for Rotating Machinery



OpenFOAM v2.3 and 2.4
foam-extend-3.1





Keywords :

- OpenFOAM
- SRF
- MRF
- cyclicAMI
- Sliding Interface
- Mixing Plane

Fumiya Nozaki

Last Updated: 2 August 2015

"This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD® trade marks."

This document covers

how to set up CFD simulation for a rotating machinery using OpenFOAM.

In particular,

- **Single Rotating Frame** (SRF)
- **Multiple Reference Frame** (MRF)
- **Arbitraly Mesh Interface** (AMI)

are and will be described in detail.

Chapter 1 Introduction

In this chapter we shall describe the classification of the methods for solving a flow around a rotating machinery using OpenFOAM.

What we need to decide

Our simulation begins with deciding the following conditions:

1. Computational Domain

2. Time Dependency

What we need to decide

Our simulation begins with deciding the following conditions:

1. Computational Domain

Does our model include some **stationary** regions or not?

2. Time Dependency

Our simulation begins with deciding the following conditions:

1. Computational Domain

2. Time Dependency

Which do we want to obtain, a **steady** or **transient** solution?

1. Computational Domain

2. Time Dependency

	Only Rotating Region	Include Stationary Region(s)
Steady	**SRFSimpleFoam** Chapter 2	**SimpleFoam + fvOptions** Chapter 3
Transient	**SRFPimpleFoam** Chapter 2	**pimpleDyMFoam** Chapter 4

Chapter 2 Single Rotating Frame

In this chapter we shall describe how to set up the solvers

SRFSimpleFoam and SRFPimpleFoam

to simulate an incompressible flow field
in a single rotating frame.

The Single Rotating Frame (SRF) model computes fluid flow
in a rotating frame of reference that is adhere to a rotating machinery.

- Steady-state solutions are possible
- Without any mesh motion

OpenFOAM solvers using Single Rotating Frame (SRF) model

Steady-state solver

SRFSimpleFoam

Transient solver

SRFPimpleFoam

- Governing Equations of SRFSimpleFoam

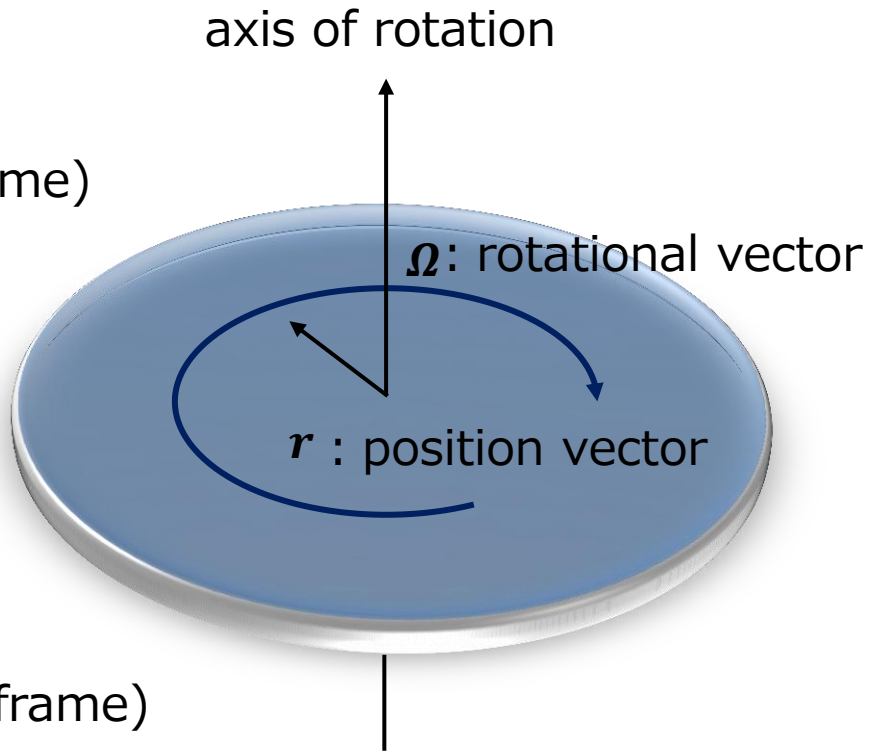
Coriolis force Centrifugal force

$$\begin{cases} \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_R) + \boxed{2\boldsymbol{\Omega} \times \boldsymbol{u}_R} + \boxed{\boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r})} = -\nabla p + \nabla \cdot \left(\nu_{eff}(\nabla \boldsymbol{u}_R + (\nabla \boldsymbol{u}_R)^T)\right) \\ \nabla \cdot \boldsymbol{u}_R = 0 \end{cases}$$

Relative velocity
(the velocity viewed from the rotating frame)

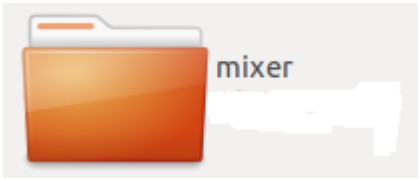
$$\boldsymbol{u}_I = \boldsymbol{u}_R + \boldsymbol{\Omega} \times \boldsymbol{r}$$

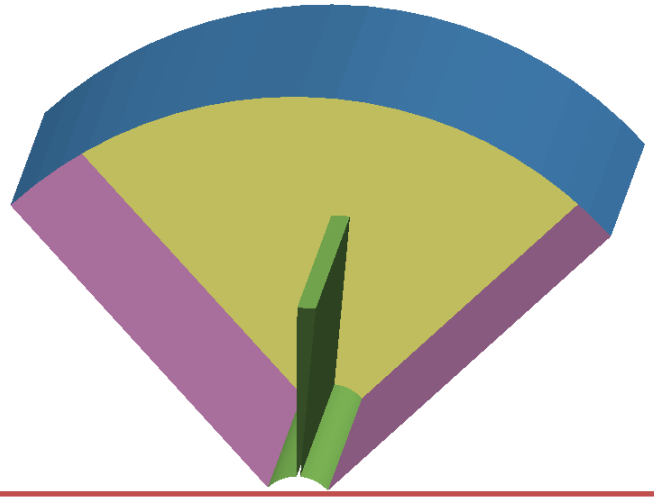
Absolute velocity
(the velocity viewed from the stationary frame)



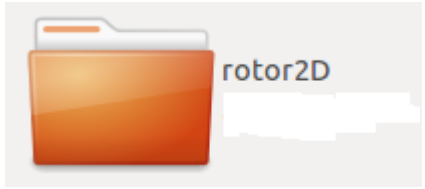
- Tutorials of SRFSimpleFoam and SRFPimpleFoam

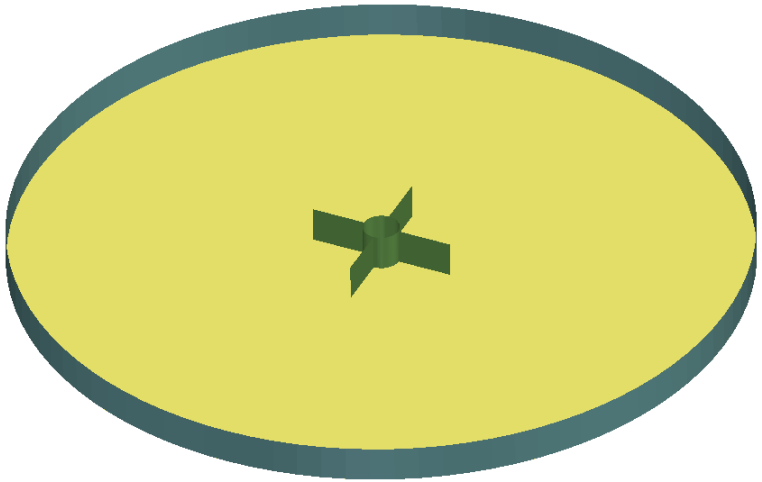
- SRFSimpleFoam





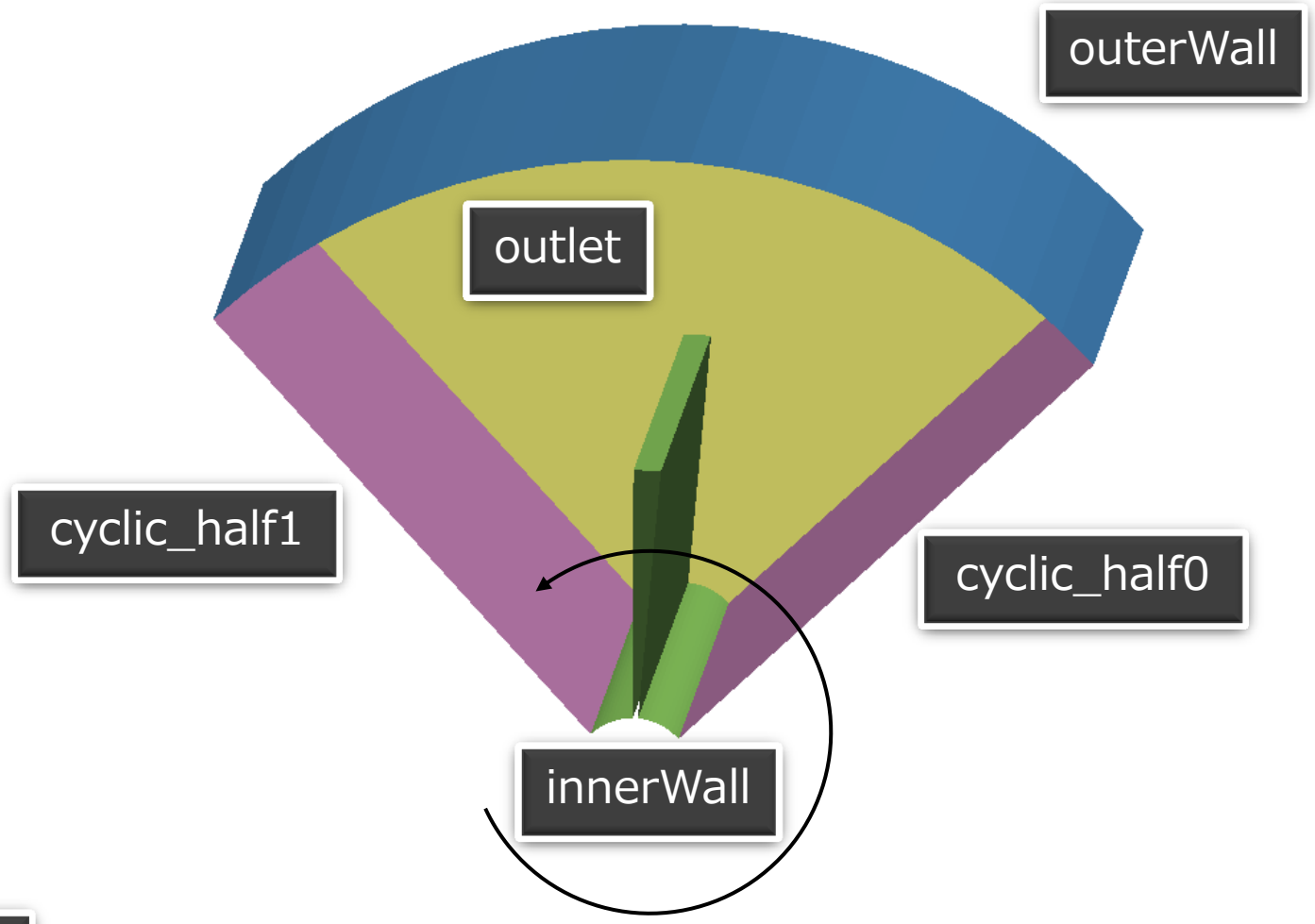
- SRFPimpleFoam





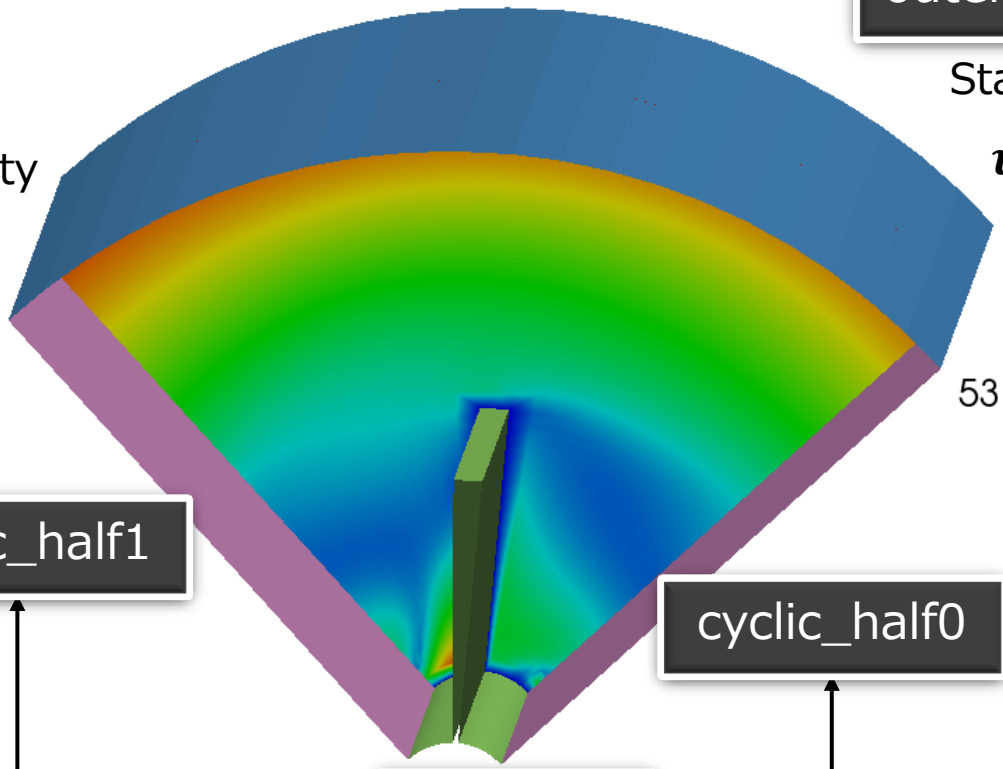
- tutorials/incompressible/SRFSimpleFoam/mixer

A blade named “innerWall” is rotating counterclockwise at 5000 rpm.

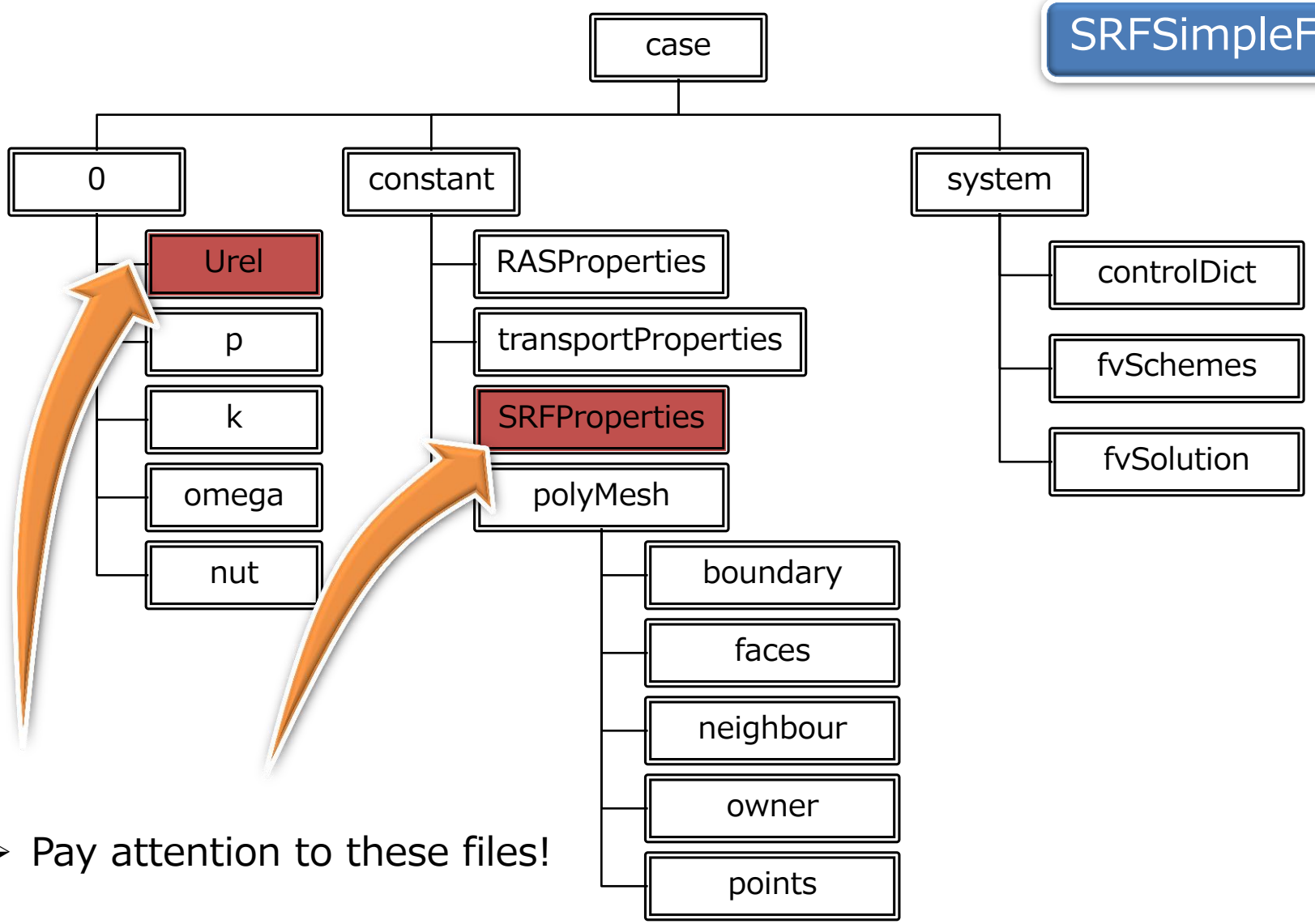


inlet patch is hidden for facilitating visualization.

- Boundary conditions of velocity field



SRFSimpleFoam



- Pay attention to these files!
  - ***Urel*** represents the relative velocity field
  - Rotating velocity condition is described in ***SRFProperties*** file

- Angular velocity condition is described in ***SRFProperties*** file.

```
/*--------------------------------*- C++ -*----------------------------------*¥
| =========                 |                                                 |
| ¥¥ / F ield               | OpenFOAM: The Open Source CFD Toolbox           |
| ¥¥ / O peration           | Version: 2.3.0                                  |
| ¥¥ / A nd                 | Web: www.OpenFOAM.org                           |
| ¥¥/ M anipulation         |                                                 |
¥*---------------------------------------------------------------------------*/
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    location "constant";
    object SRFProperties;
}
// * * * * * * * * *

SRFModel rpm;

axis ( 0 0 1 );

rpmCoeffs
{
    rpm 5000;
}


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

Angular velocity is specified in terms of revolutions-per-minute [rpm]

Axis of rotation

5000 rpm $\Rightarrow 5000 * 2\pi/60 = 523.6$ [rad/s]

Rotational direction

When the axis vector points to you

- rpm>0 ⇒ rotation in **counterclockwise** direction
- rpm<0 ⇒ rotation in **clockwise** direction

- In this tutorial, ***SRFVelocity*** boundary condition is used on inlet and outerWall boundaries.

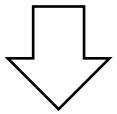
```
inlet
{
    type            SRFVelocity;
    inletValue      uniform (0 0 -10);
    relative        no;
    value           uniform (0 0 0);
}
```

This means $\boldsymbol{u}_I = (0\ 0\ -10)$.

```
outerWall
{
    type            SRFVelocity;
    inletValue      uniform (0 0 0);
    relative        no;
    value           uniform (0 0 0);
}
```

This means $\boldsymbol{u}_I = \mathbf{0}$ (a stationary wall).

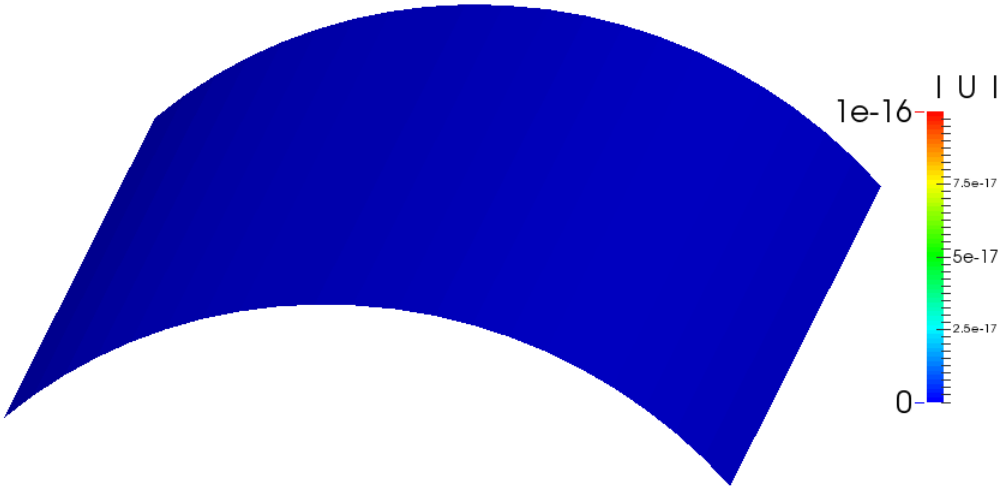
By setting ***relative*** up as ***no***



Boundary condition of
the relative velocity $\boldsymbol{u}_r$ (Urel)
is able to be specified
in terms of the absolute velocity.

Closer look at the velocity on outerWall

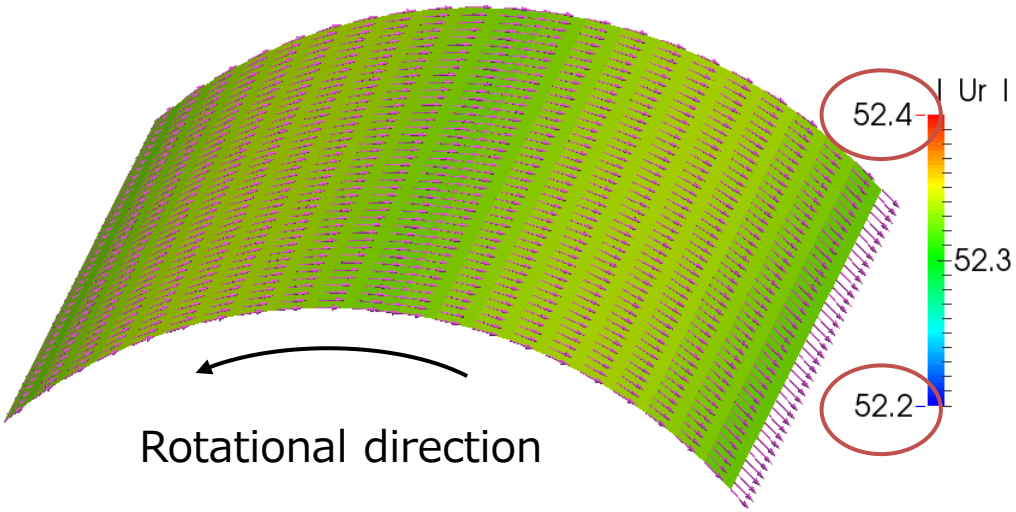
In the stationary coordinate system



$$\boldsymbol{u}_I = \mathbf{0}$$

In the rotating coordinate system

Vectors show $\boldsymbol{u}_r$



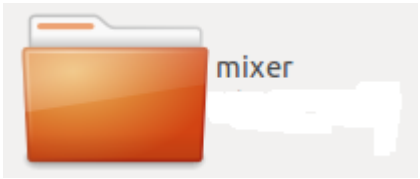
Radius of outerWall is 0.1 m

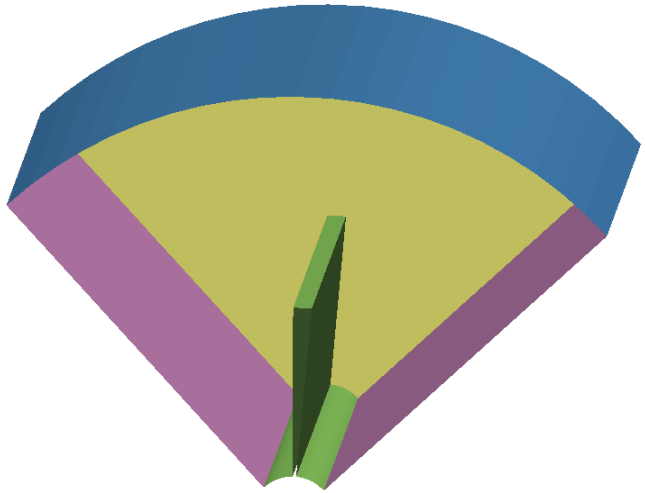
$$|\boldsymbol{u}_r| = |\boldsymbol{u}_I - \boldsymbol{\Omega} \times \boldsymbol{r}|$$

$$= 523.6 \cdot 0.1 = 52.36\ [m/s]$$

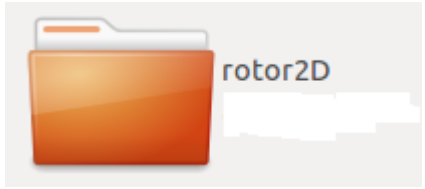
- Tutorials of SRFSimpleFoam and SRFPimpleFoam

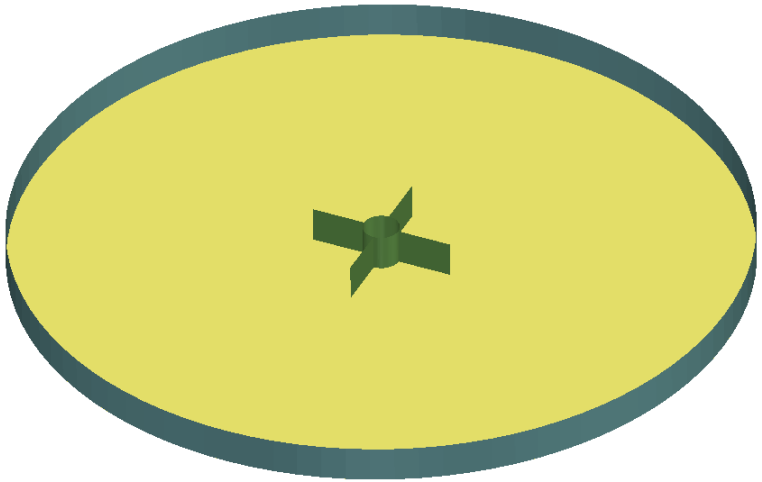
  - SRFSimpleFoam





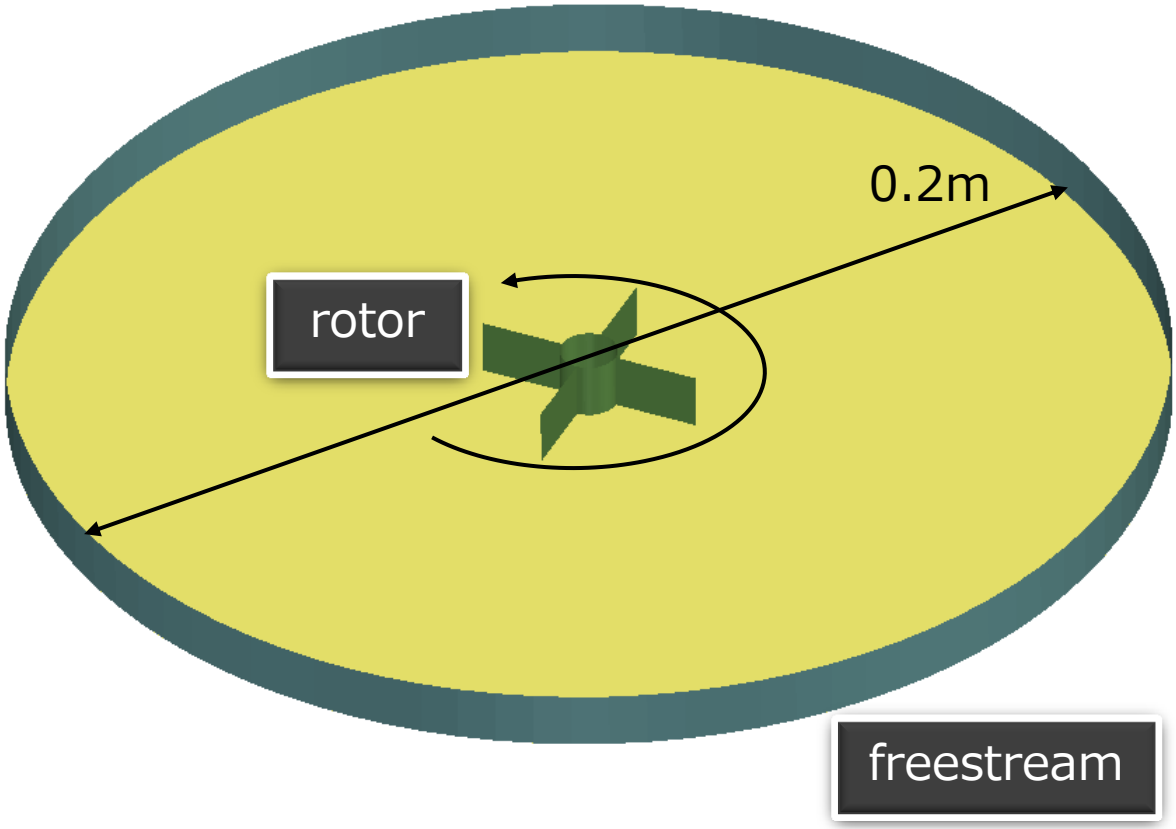
  - SRFPimpleFoam



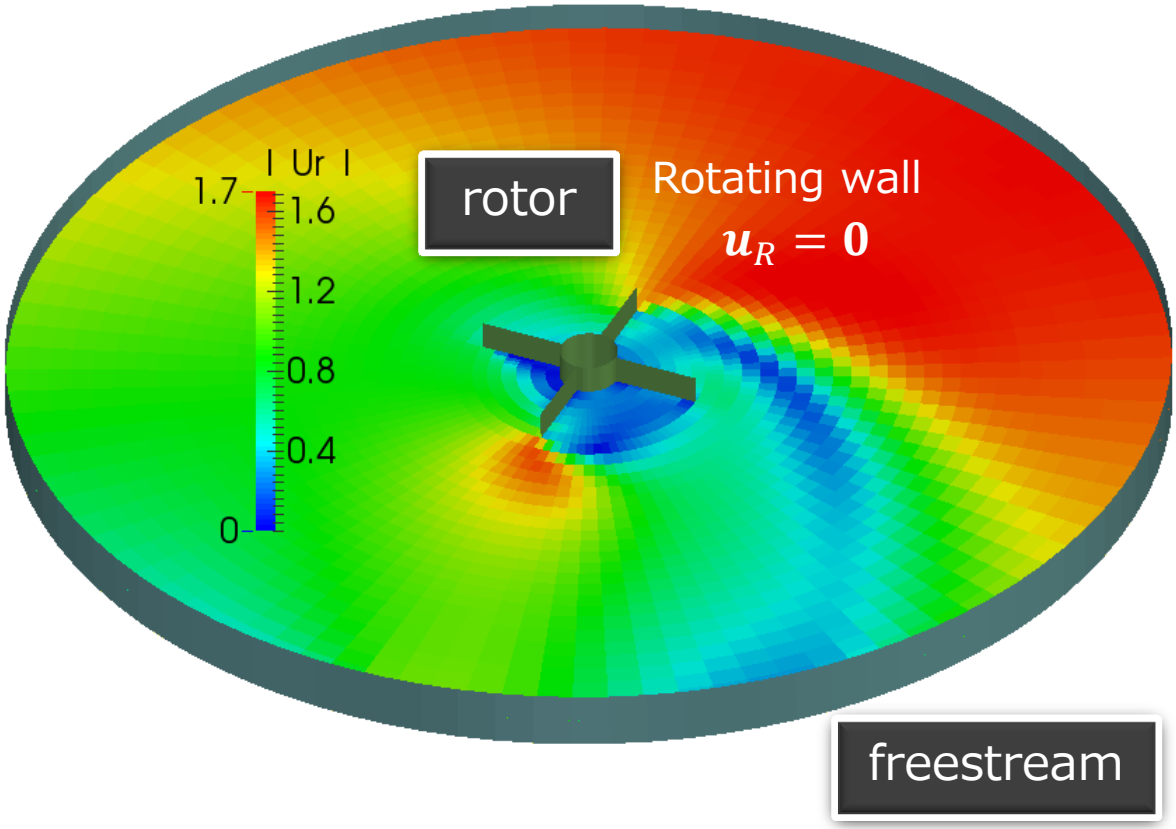


- tutorials/incompressible/SRFPimpleFoam/rotor2D

A wall named “rotor” is rotating counterclockwise at 60 rpm.



- Boundary conditions of velocity field



- Calculation of the additional force terms

$$\begin{cases} (\boldsymbol{u}_R \cdot \nabla)\boldsymbol{u}_R + 2\boldsymbol{\Omega} \times \boldsymbol{u}_R + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r}) = -\nabla p + \nabla \cdot \left(\nu_{eff}(\nabla \boldsymbol{u}_R + (\nabla \boldsymbol{u}_R)^T)\right) \\ \nabla \cdot \boldsymbol{u}_R = 0 \end{cases}$$

UrelEqn.H

```
// Relative momentum predictor


tmp<fvVectorMatrix> UrelEqn
(
    fvm::div(phi, Urel)
  + turbulence->divDevReff(Urel)
  + SRF->Su()
 ==
    fvOptions(Urel)
);
```

SRFModel.C

```
Foam::tmp<Foam::DimensionedField<Foam::vector, Foam::volMesh> >
Foam::SRF::SRFModel::Su() const
{
    return Fcoriolis() + Fcentrifugal();
}
```

```
2.0*omega_ ^ Urel_
```

```
omega_ ^ (omega_ ^ (mesh_.C() - origin_))
```

- Calculation of the absolute velocity (Uabs)

SRFSimpleFoam.C

```
Urel + SRF->U()
```

```
omega_ ^ ((mesh_.C() - origin_) - axis_*(axis_ &
                (mesh_.C() - origin_)))
```

Implementation | SRFVelocity

```
void Foam::SRFVelocityFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    // If not relative to the SRF include the effect of the SRF
    if (!relative_)
    {
        // Get reference to the SRF model
        const SRF::SRFModel& srf =
            db().lookupObject<SRF::SRFModel>("SRFProperties");

        // Determine patch velocity due to SRF
        const vectorField SRFVelocity(srf.velocity(patch().Cf()));

        operator==(-SRFVelocity + inletValue_);
    }
    // If already relative to the SRF simply supply the inlet value as a fixed
    // value
    else
    {
        operator==(inletValue_);
    }

    fixedValueFvPatchVectorField::updateCoeffs();
}
```

src/finiteVolume/cfdTools/general/SRF/derivedFvPatchFields/SRFVelocityFvPatchVectorFiel/SRFVelocityFvPatchVectorField.C

If "**relative**" is "**no**"

Calculate $\Omega \times r$

SRFModel.C l.60

$u_R = -\Omega \times r + u_I$

If "**relative**" is "**yes**"

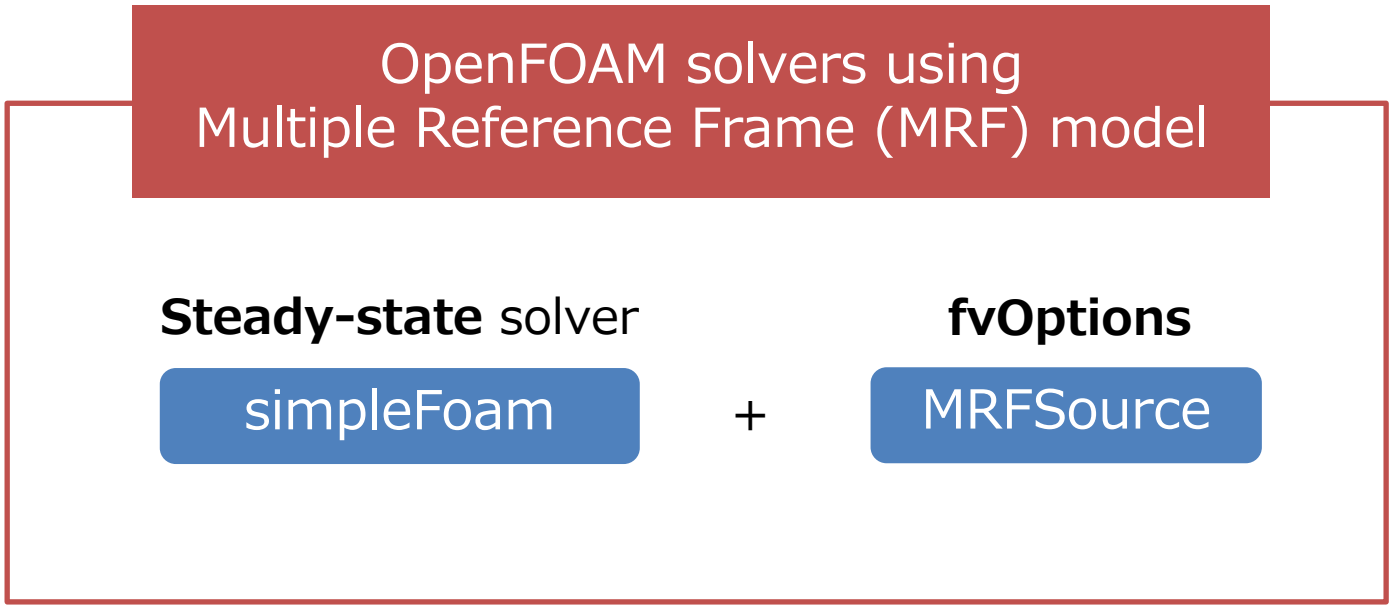
Chapter 3 Multiple Reference Frame

In this chapter we shall describe how to set up
the multiple reference frame simulation in OpenFOAM.
This model is activated using
“fvOptions” functionality.

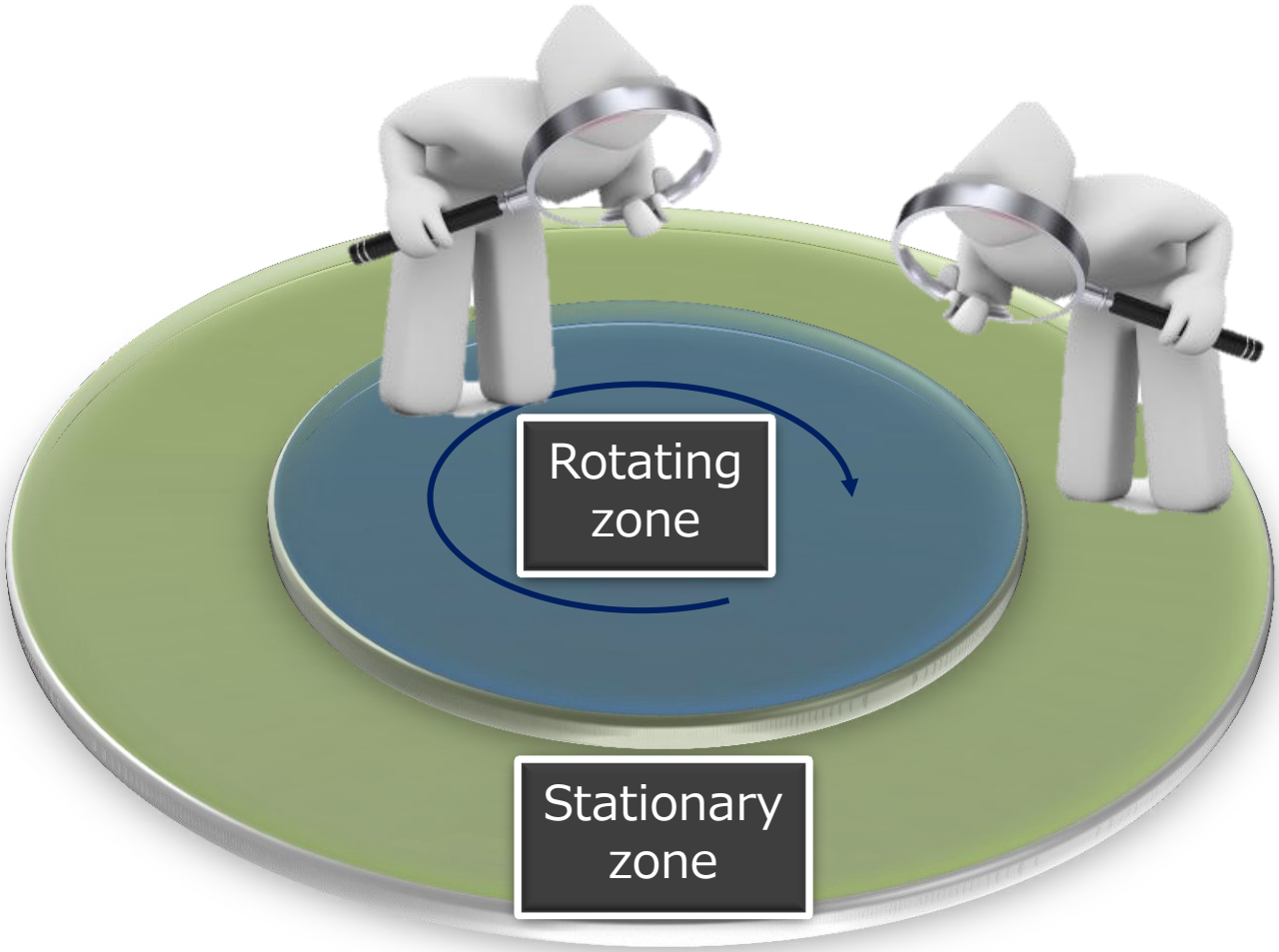
The Multiple Reference Frame (SRF) model computes fluid flow using both the rotating and stationary reference frames.

- Rotating zone is solved in the rotating frame
- Stationary zone is solved in the stationary frame

} Multiple frames



Conceptual image



$$\boxed{\nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_R)} = \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) - \nabla \cdot \{\boldsymbol{u}_R (\boldsymbol{\Omega} \times \boldsymbol{r})\}$$

Sometimes denoted $\boldsymbol{u}_R \otimes \boldsymbol{u}_R$

$$\nabla \cdot \{\boldsymbol{u}_R (\boldsymbol{\Omega} \times \boldsymbol{r})\} = \nabla \cdot \begin{pmatrix} u_{R_1}(\Omega_2 r_3 - \Omega_3 r_2) & u_{R_1}(\Omega_3 r_1 - \Omega_1 r_3) & u_{R_1}(\Omega_1 r_2 - \Omega_2 r_1) \\ u_{R_2}(\Omega_2 r_3 - \Omega_3 r_2) & u_{R_2}(\Omega_3 r_1 - \Omega_1 r_3) & u_{R_2}(\Omega_1 r_2 - \Omega_2 r_1) \\ u_{R_3}(\Omega_2 r_3 - \Omega_3 r_2) & u_{R_3}(\Omega_3 r_1 - \Omega_1 r_3) & u_{R_3}(\Omega_1 r_2 - \Omega_2 r_1) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial}{\partial x_1}\{u_{R_1}(\Omega_2 r_3 - \Omega_3 r_2)\} + \frac{\partial}{\partial x_2}\{u_{R_2}(\Omega_2 r_3 - \Omega_3 r_2)\} + \frac{\partial}{\partial x_3}\{u_{R_3}(\Omega_2 r_3 - \Omega_3 r_2)\} \\ \frac{\partial}{\partial x_1}\{u_{R_1}(\Omega_3 r_1 - \Omega_1 r_3)\} + \frac{\partial}{\partial x_2}\{u_{R_2}(\Omega_3 r_1 - \Omega_1 r_3)\} + \frac{\partial}{\partial x_3}\{u_{R_3}(\Omega_3 r_1 - \Omega_1 r_3)\} \\ \frac{\partial}{\partial x_1}\{u_{R_1}(\Omega_1 r_2 - \Omega_2 r_1)\} + \frac{\partial}{\partial x_2}\{u_{R_2}(\Omega_1 r_2 - \Omega_2 r_1)\} + \frac{\partial}{\partial x_3}\{u_{R_3}(\Omega_1 r_2 - \Omega_2 r_1)\} \end{pmatrix}$$

$$= \begin{pmatrix} (\Omega_2 r_3 - \Omega_3 r_2)\nabla \cdot \boldsymbol{u}_R \\ (\Omega_3 r_1 - \Omega_1 r_3)\nabla \cdot \boldsymbol{u}_R \\ (\Omega_1 r_2 - \Omega_2 r_1)\nabla \cdot \boldsymbol{u}_R \end{pmatrix} + \begin{pmatrix} \boldsymbol{u}_R \cdot \nabla(\Omega_2 r_3 - \Omega_3 r_2) \\ \boldsymbol{u}_R \cdot \nabla(\Omega_3 r_1 - \Omega_1 r_3) \\ \boldsymbol{u}_R \cdot \nabla(\Omega_1 r_2 - \Omega_2 r_1) \end{pmatrix} = \boldsymbol{\Omega} \times \boldsymbol{u}_R$$

$$\nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_R) = \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) - \boldsymbol{\Omega} \times \boldsymbol{u}_R$$

- In the rotating frame

$$
\begin{aligned}
&\nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_R) + 2\boldsymbol{\Omega} \times \boldsymbol{u}_R + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r}) \\
&= \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) - \boldsymbol{\Omega} \times \boldsymbol{u}_R + 2\boldsymbol{\Omega} \times \boldsymbol{u}_R + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r}) \\
&= \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) + \boldsymbol{\Omega} \times \boldsymbol{u}_R + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r}) \\
&= \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) + \boldsymbol{\Omega} \times \{\boldsymbol{u}_R + (\boldsymbol{\Omega} \times \boldsymbol{r})\} \\
&= \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) + \boldsymbol{\Omega} \times \boldsymbol{u}_I \quad (1)
\end{aligned}
$$

- Governing equations of SRF can be rearranged in the following manner:

$$
\begin{cases}
\nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_R) + 2\boldsymbol{\Omega} \times \boldsymbol{u}_R + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r}) = -\nabla p + \nabla \cdot \left(\nu_{eff} (\nabla \boldsymbol{u}_R + (\nabla \boldsymbol{u}_R)^T)\right) \\
\nabla \cdot \boldsymbol{u}_R = 0
\end{cases}
$$

$$
\begin{cases}
\nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) + \boldsymbol{\Omega} \times \boldsymbol{u}_I = -\nabla p + \nabla \cdot \left(\nu_{eff} (\nabla \boldsymbol{u}_I + (\nabla \boldsymbol{u}_I)^T)\right) \\
\nabla \cdot \boldsymbol{u}_R = 0
\end{cases}
$$

- Governing equations [1]

Rotating zone

1 Coriolis force

$$\begin{cases} \nabla \cdot (\boldsymbol{u}_R \boldsymbol{u}_I) + \boldsymbol{\Omega} \times \boldsymbol{u}_I = -\nabla p + \nabla \cdot \left(\nu_{eff} (\nabla \boldsymbol{u}_I + (\nabla \boldsymbol{u}_I)^T) \right) \\ \nabla \cdot \boldsymbol{u}_R = 0 \end{cases}$$

2

Stationary zone

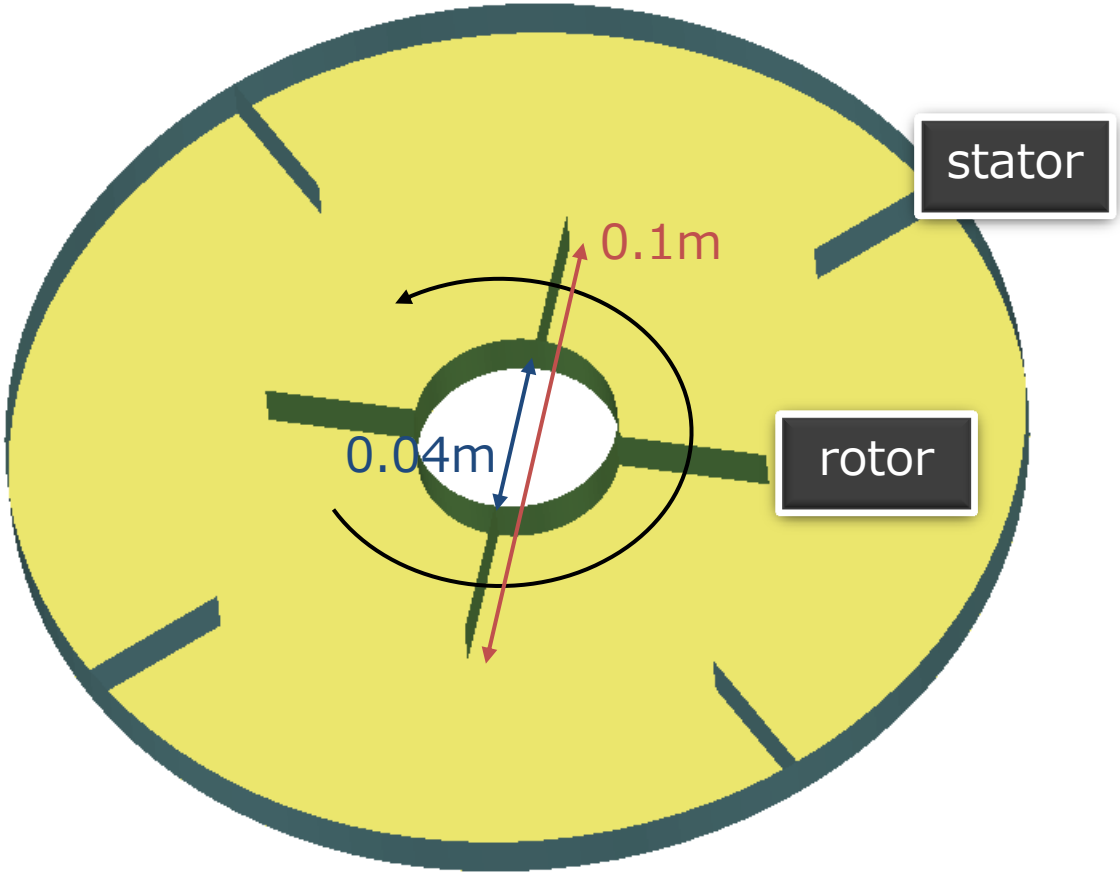
$$\begin{cases} \nabla \cdot (\boldsymbol{u}_I \boldsymbol{u}_I) = -\nabla p + \nabla \cdot \left(\nu_{eff} (\nabla \boldsymbol{u}_I + (\nabla \boldsymbol{u}_I)^T) \right) \\ \nabla \cdot \boldsymbol{u}_I = 0 \end{cases}$$

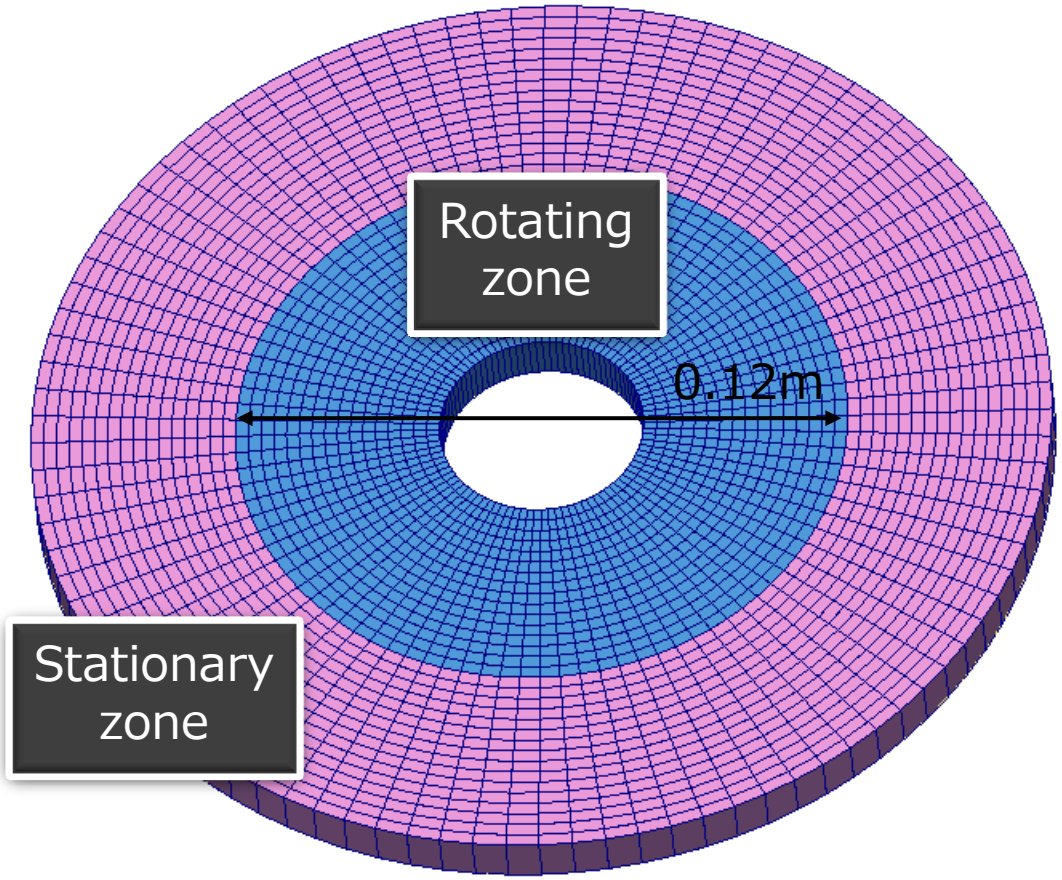
- In each rotating zone

1. the Coriolis force is added to the governing equations
2. the flux is calculated from the relative velocity $\boldsymbol{u}_R$

- tutorials/incompressible/simpleFoam/mixerVessel2D

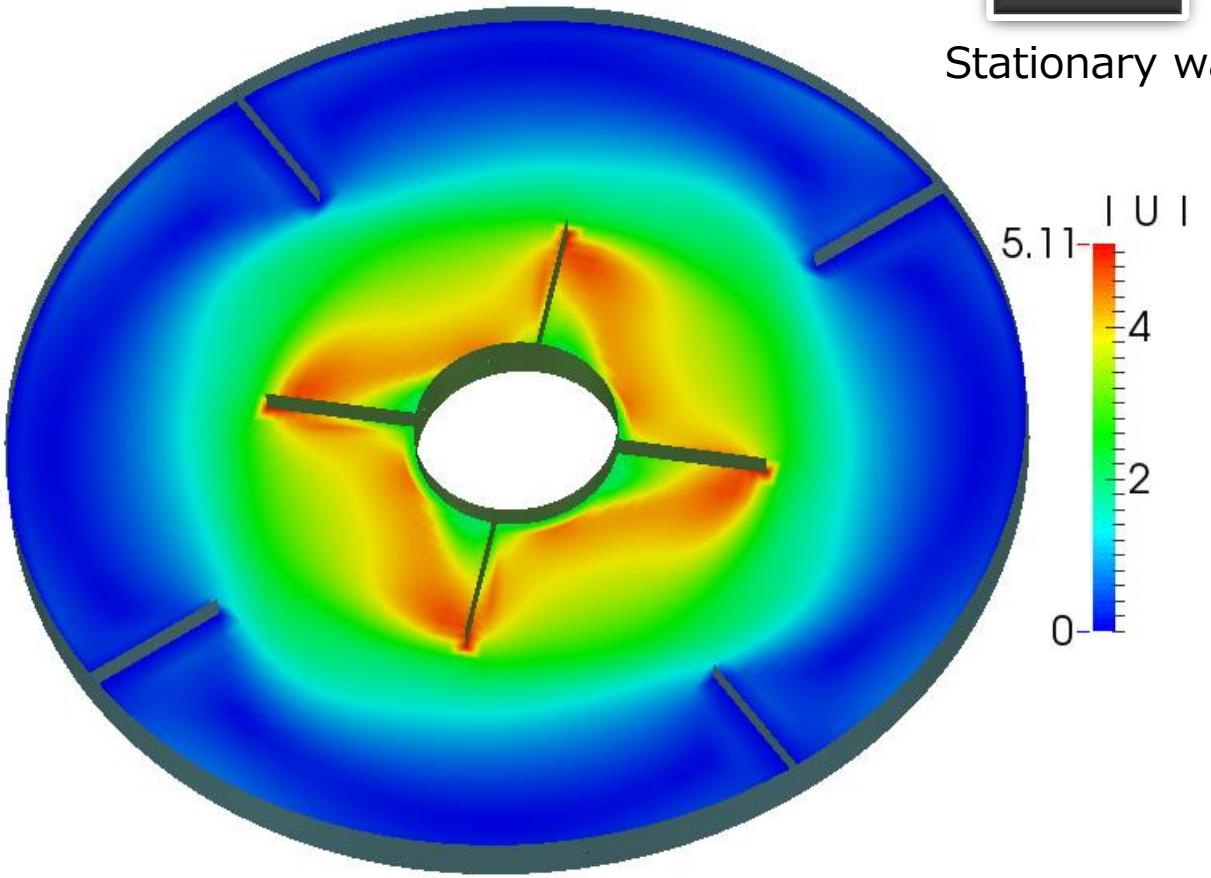
A wall named “rotor” is rotating counterclockwise at 104.72 rad/s.




Rotating zone
0.12m
Stationary zone

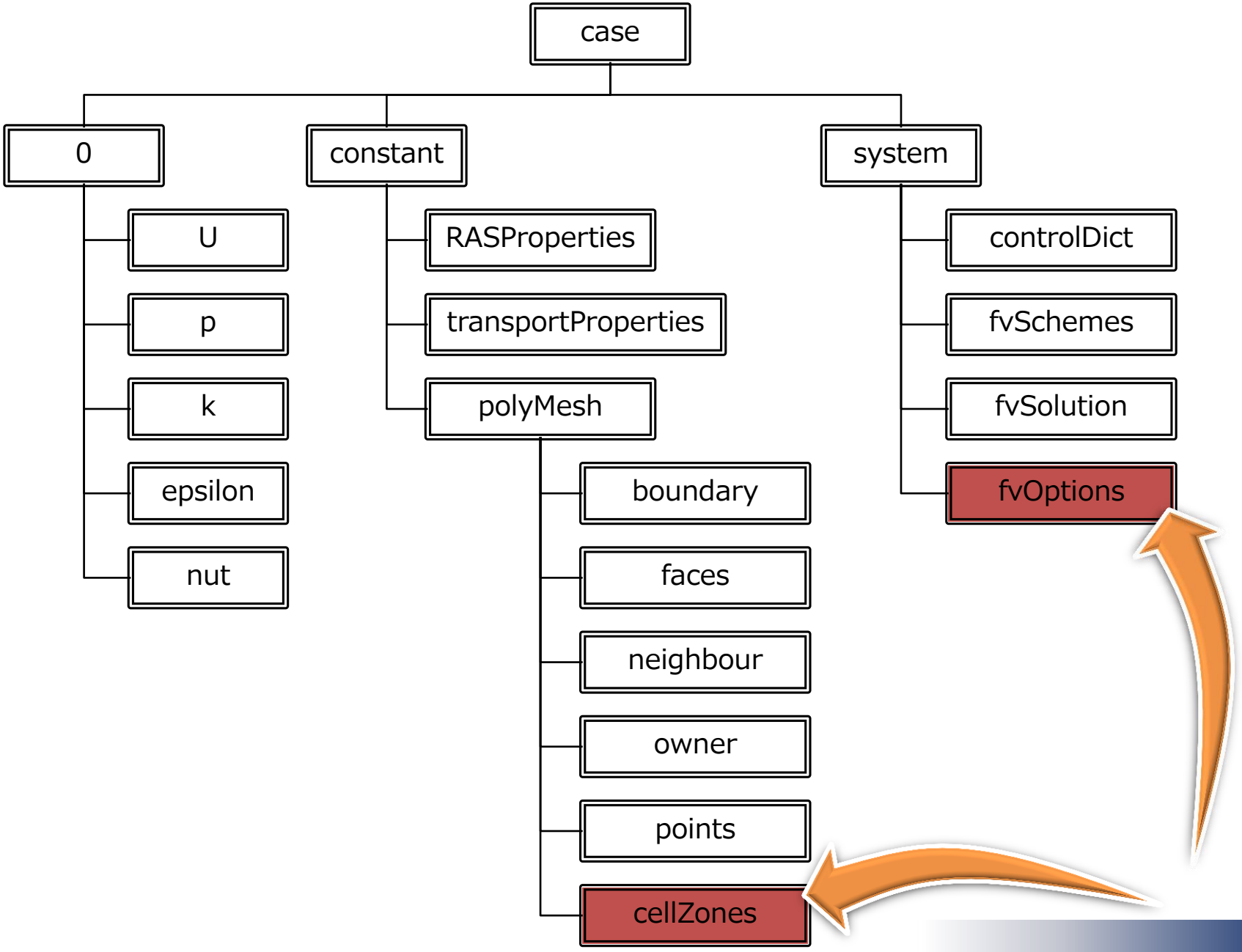
stator

Stationary wall



rotor

Rotating wall


case
0
constant
system
U
p
k
epsilon
nut
RASProperties
transportProperties
polyMesh
boundary
faces
neighbour
owner
points
cellZones
controlDict
fvSchemes
fvSolution
fvOptions

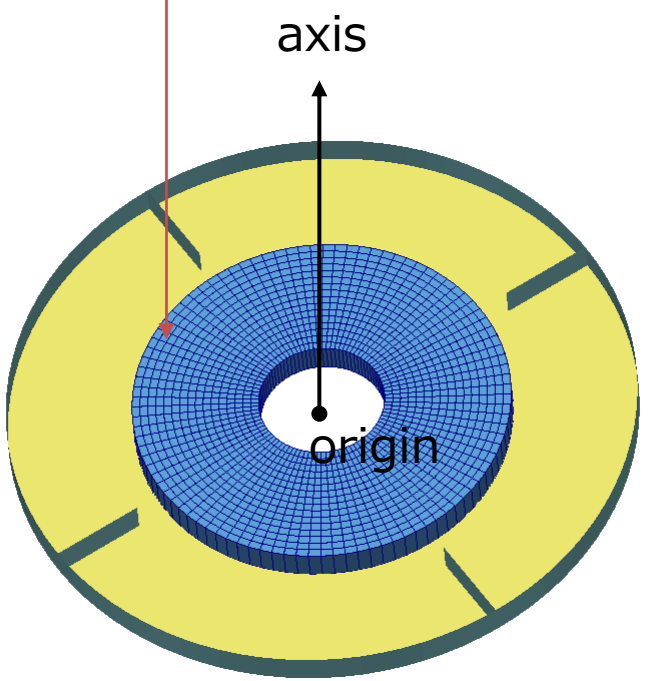
fvOptions Description

system/fvOptions

```
MRF1
{
    type            MRFSource;
    active          ture;
    selectionMode   cellZone;
    cellZone        rotor;

    MRFSourceCoeffs
    {
        nonRotatingPatches ();
        origin          (0 0 0);
        axis            (0 0 1);
        omega           104.72;
    }
}
```

Unit: [rad/s]

Specifying the rotating zone by cellZone named rotor.



Following options are available for "selectionMode"

- all
- cellSet
- cellZone
- mapRegion
- points

cellZones Description

constant/polyMesh/cellZones

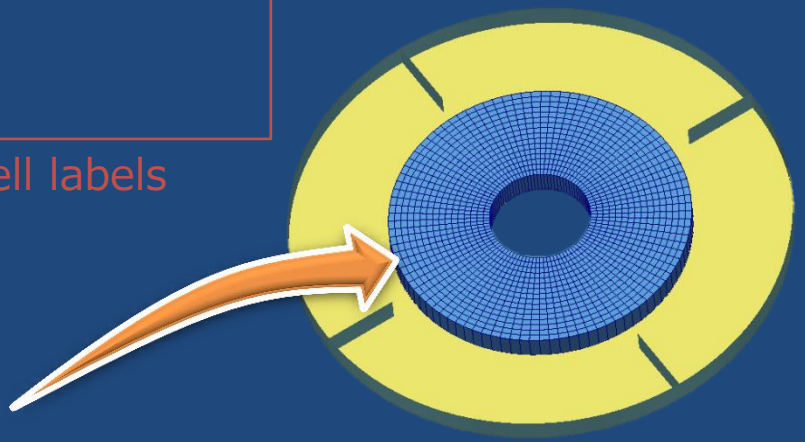
```
FoamFile
{
    version     2.0;
    format      ascii;
    class       regIOobject;
    location    "constant/polyMesh";
    object      cellZones;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

1
(
rotor
{
    type cellZone;
cellLabels      List<label>
1536
(
0
1
2
3
          (snip)
1533
1534
1535
)
;
}
)

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

The number of cellZones

The name of cellZones

The number of cells
that make up cellZones “rotor”

List of their cell labels

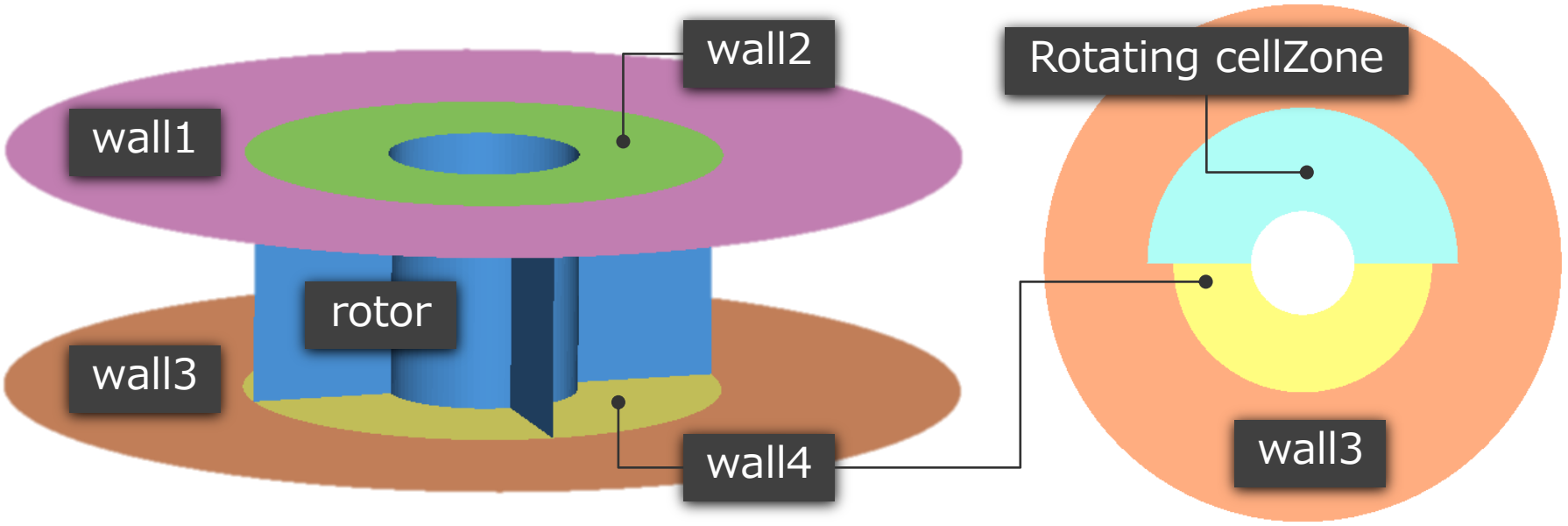


```
MRFSourceCoeffs
{
    nonRotatingPatches ();
    origin          (0 0 0);
    axis            (0 0 1);
    omega           104.72;
}
```

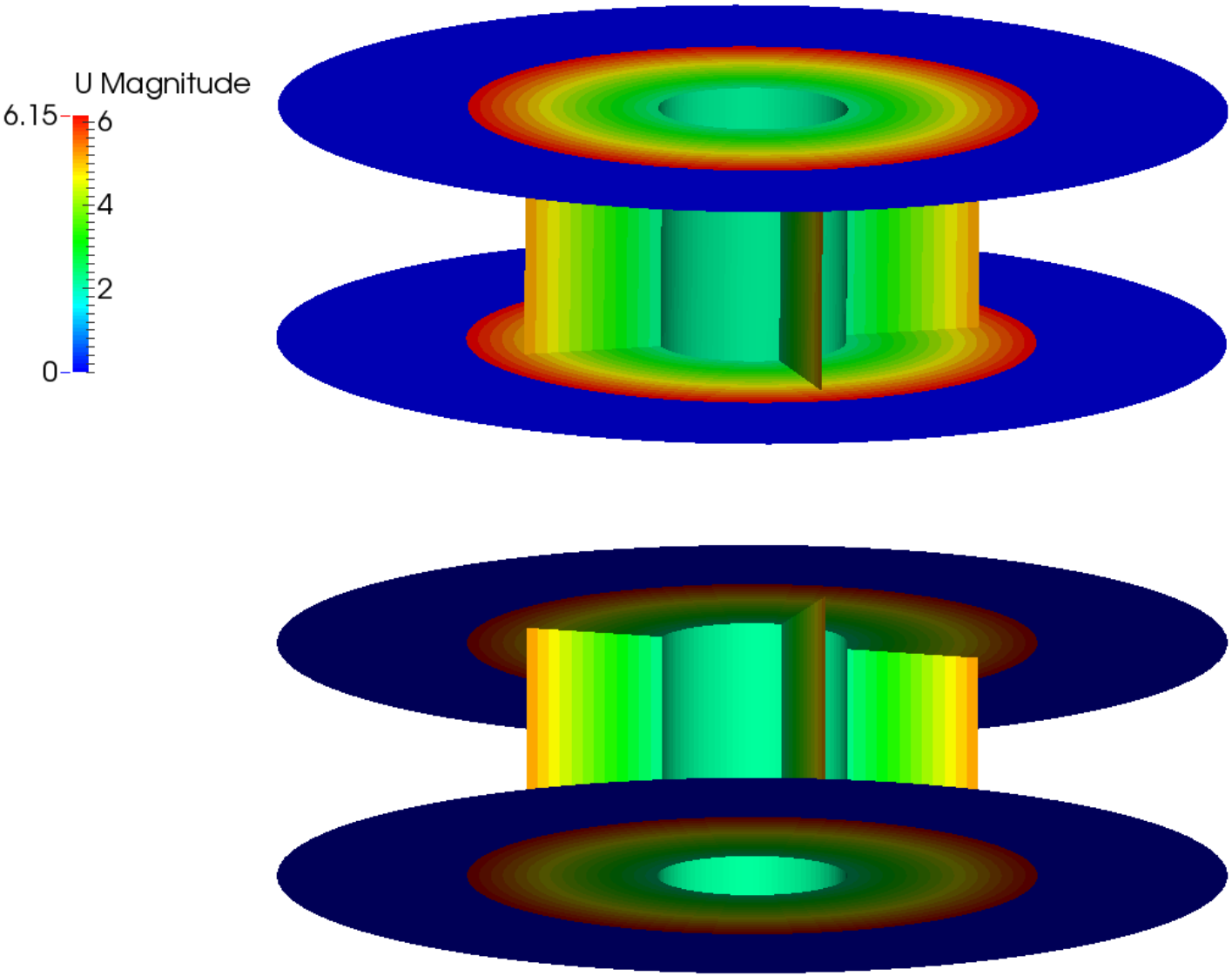
stationary patches whose adjacent (owner) cell is in a rotating cell zone.

We’ll compare the following two settings:

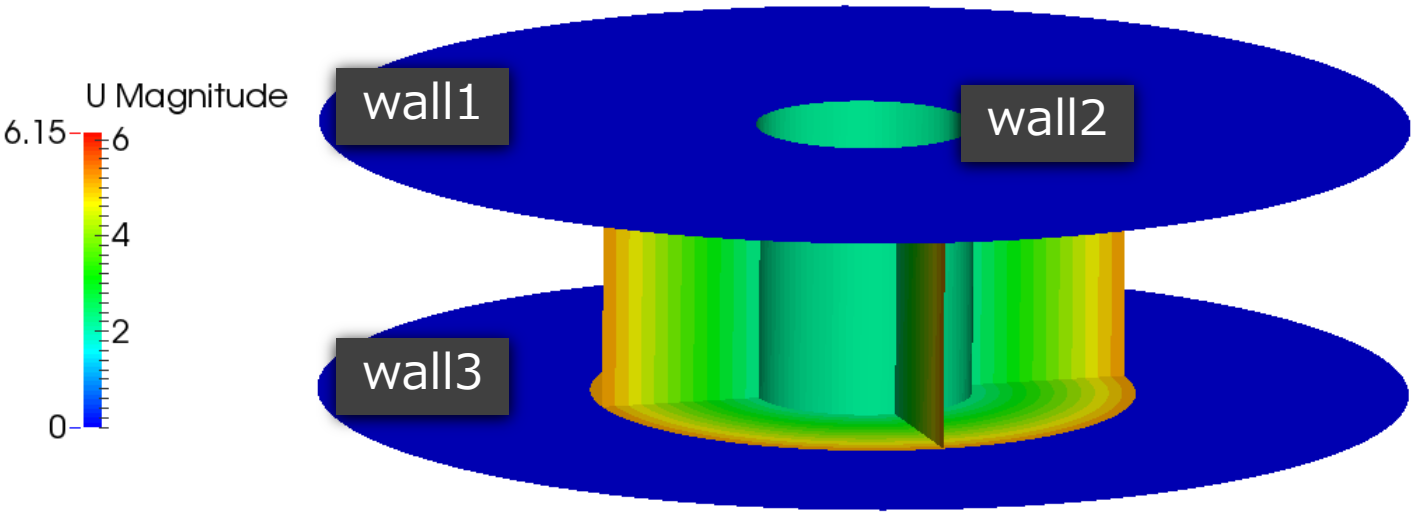
- Case1: nonRotatingPatches ();
- Case2: nonRotatingPatches (wall1 wall2 wall3);



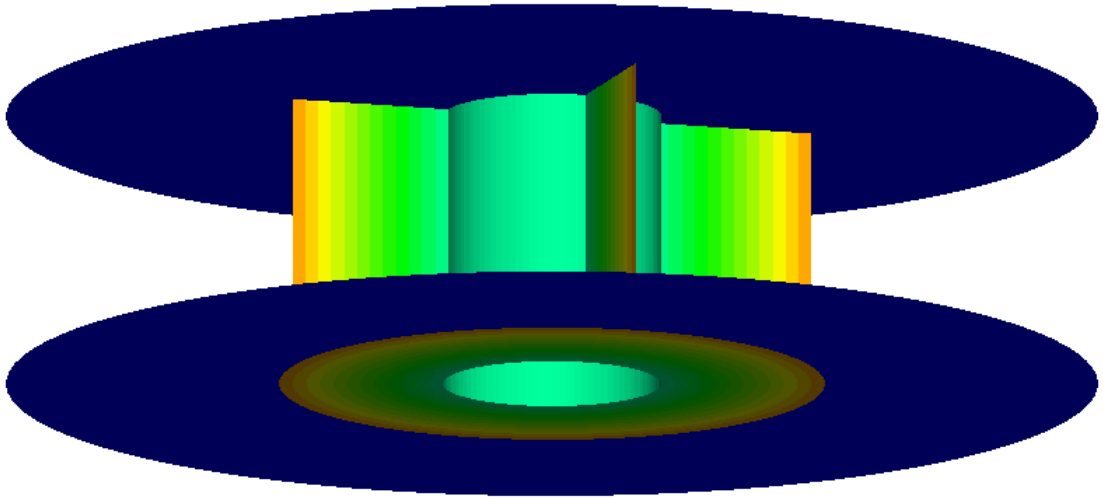
Case1: nonRotatingPatches ();



Case2: nonRotatingPatches (wall1 wall2 wall3);



wall1, wall2 and wall3 are treated as stationary patches.



Implementation | Related source code

- src/fvOptions/sources/derived/MRFSource
  - MRFSource.C
  - MRFSource.H

- src/finiteVolume/cfdTools/general/MRF
  - IOMRFZoneList.C
  - IOMRFZoneList.H
  - MRFZone.C
  - MRFZone.H
  - MRFZoneI.H
  - MRFZoneList.C
  - MRFZoneList.H
  - MRFZoneTemplates.C

- Add the Coriolis force to the right-hand side of the momentum equation

simpleFoam/UEqn.H

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

```
Foam::tmp<Foam::fvMatrix<Type> > Foam::fv::optionList::operator()
(
    GeometricField<Type, fvPatchField, volMesh>& fld,
    const word& fieldName
)
{
    (snip)
    source.addSup(mtx, fieldI);
    (snip)
}
```

src/fvOptions/sources/derived/MRFSource/MRFSource.C

```
void Foam::fv::MRFSource::addSup
(
    fvMatrix<vector>& eqn,
    const label fieldI
)
{
    // Update the velocity boundary conditions for changes in rotation speed
    mrfPtr_->correctBoundaryVelocity(const_cast<volVectorField&>(eqn.psi()));

    // Add to rhs of equation
    mrfPtr_->addCoriolis(eqn, true);
}
```

Implementation | Calculation of Coriolis force

src/finiteVolume/cfdTools/general/MRF/MRFZone.C

```
void Foam::MRFZone::addCoriolis(fvVectorMatrix& UEqn, const bool rhs) const
{
    if (cellZoneID_ == -1)
    {
        return;
    }

    const labelList& cells = mesh_.cellZones()[cellZoneID_];
    const scalarField& V = mesh_.V();
    vectorField& Usource = UEqn.source();
    const vectorField& U = UEqn.psi();

    const vector Omega = this->Omega();

    if (rhs)
    {
        forAll(cells, i)
        {
            label celli = cells[i];
            Usource[celli] += V[celli]*(Omega ^ U[celli]);
        }
    }
    else
    {
        forAll(cells, i)
        {
            label celli = cells[i];
            Usource[celli] -= V[celli]*(Omega ^ U[celli]);
        }
    }
}
```

List of cell labels included in the cell zone

Adding the Coriolis force $-\boldsymbol{\Omega} \times \boldsymbol{u}$ to the right-hand side as a source term

- Classify all faces into <u>three</u> groups (faceType = 0, 1, 2)
  - For internal faces

src/finiteVolume/cfdTools/general/MRF/MRFZone.C

```
void Foam::MRFZone::setMRFFaces()
{
    const polyBoundaryMesh& patches = mesh_.boundaryMesh();

    labelList faceType(mesh_.nFaces(), 0);

    const labelList& own = mesh_.faceOwner();
    const labelList& nei = mesh_.faceNeighbour();

    // Cells in zone
    boolList zoneCell(mesh_.nCells(), false);

    if (cellZoneID_ != -1)
    {
        const labelList& cellLabels = mesh_.cellZones()[cellZoneID_];
        forAll(cellLabels, i)
        {
            zoneCell[cellLabels[i]] = true;
        }
    }


    label nZoneFaces = 0;

    for (label faceI = 0; faceI < mesh_.nInternalFaces(); faceI++)
    {
        if (zoneCell[own[faceI]] || zoneCell[nei[faceI]])
        {
            faceType[faceI] = 1;
            nZoneFaces++;
        }
    }
```

faceType = 1

Either owner or neighbour cell is in the cell zone

Implementation | Calculation of face flux

- For boundary faces

```
labelHashSet excludedPatches(excludedPatchLabels_);

forAll(patches, patchI)
{
    const polyPatch& pp = patches[patchI];

    if (pp.coupled() || excludedPatches.found(patchI))
    {
        forAll(pp, i)
        {
            label faceI = pp.start()+i;

            if (zoneCell[own[faceI]])
            {
                faceType[faceI] = 2;
                nZoneFaces++;
            }
        }
    }
    else if (!isA<emptyPolyPatch>(pp))
    {
        forAll(pp, i)
        {
            label faceI = pp.start()+i;

            if (zoneCell[own[faceI]])
            {
                faceType[faceI] = 1;
                nZoneFaces++;
            }
        }
    }
}
```

faceType = 2

Coupled patch face or excluded patch face whose owner cell is in the cell zone

faceType = 1

Normal patch face whose owner cell is in the cell zone

- Create two lists of face labels per patch

```
//- Outside faces (per patch) that move with the MRF
labelListList includedFaces_;

//- Excluded faces (per patch) that do not move with the MRF
labelListList excludedFaces_;
```

includedFaces_[patchi][i] = local label of i^{th} face whose faceType = 1 on the patchi^{th} patch

excludedFaces_[patchi][i] = local label of i^{th} face whose faceType = 2 on the patchi^{th} patch

src/finiteVolume/cfdTools/general/MRF/MRFZone.C

```
forAll(patches, patchi)
{
    const polyPatch& pp = patches[patchi];

    forAll(pp, patchFacei)
    {
        label faceI = pp.start() + patchFacei;

        if (faceType[faceI] == 1)
        {
            includedFaces_[patchi][nIncludedFaces[patchi]++] = patchFacei;
        }
        else if (faceType[faceI] == 2)
        {
            excludedFaces_[patchi][nExcludedFaces[patchi]++] = patchFacei;
        }
    }
}
```

- Modify the face flux field

```
    surfaceScalarField phiHbyA("phiHbyA", fvc::interpolate(HbyA) & mesh.Sf());

    fvOptions.makeRelative(phiHbyA);

    adjustPhi(phiHbyA, U, p);
```

simpleFoam/pEqn.H

```
void Foam::fv::optionList::makeRelative(surfaceScalarField& phi) const
{
    forAll(*this, i)
    {
        this->operator[](i).makeRelative(phi);
    }
}
```

src/fvOptions/fvOptions/fvOptionList.C

```
void Foam::fv::MRFSource::makeRelative(surfaceScalarField& phi) const
{
    mrfPtr_->makeRelative(phi);
}
```

src/fvOptions/sources/derived/MRFSource/MRFSource.C

```
void Foam::MRFZone::makeRelative(surfaceScalarField& phi) const
{
    makeRelativeRhoFlux(geometricOneField(), phi);
}
```

src/finiteVolume/cfdTools/general/MRF/MRFZone.C

Implementation | Calculation of face flux

src/finiteVolume/cfdTools/general/MRF/MRFZoneTemplates.C

```
template<class RhoFieldType>
void Foam::MRFZone::makeRelativeRhoFlux
(
    const RhoFieldType& rho,
    surfaceScalarField& phi
) const
{
    const surfaceVectorField& Cf = mesh_.Cf();
    const surfaceVectorField& Sf = mesh_.Sf();

    const vector Omega = omega_->value(mesh_.time().timeOutputValue())*axis_;

    const vectorField& Cfi = Cf.internalField();
    const vectorField& Sfi = Sf.internalField();
    scalarField& phii = phi.internalField();

    // Internal faces
    forAll(internalFaces_, i)
    {
        label facei = internalFaces_[i];
        phii[facei] -= rho[facei]*(Omega ^ (Cfi[facei] - origin_)) & Sfi[facei];
    }

    makeRelativeRhoFlux(rho.boundaryField(), phi.boundaryField());
}
```

Modification of the internal face flux

Modification of the boundary face flux

- "internalFaces_" is a label list of the internal faces whose owner or neighbour cell is in the rotating cell zone.
- According to the governing equation, the face fluxes of these internal faces are modified by subtracting $(\boldsymbol{\Omega} \times \boldsymbol{r}) \cdot \boldsymbol{S}_f$ from the absolute face fluxes:

$$\boldsymbol{u}_R \cdot \boldsymbol{S}_f = \boldsymbol{u}_I \cdot \boldsymbol{S}_f - (\boldsymbol{\Omega} \times \boldsymbol{r}) \cdot \boldsymbol{S}_f$$

Implementation | Calculation of face flux

src/finiteVolume/cfdTools/general/MRF/MRFZoneTemplates.C

```
template<class RhoFieldType>
void Foam::MRFZone::makeRelativeRhoFlux
(
    const RhoFieldType& rho,
    FieldField<fvsPatchField, scalar>& phi
) const
{
    const surfaceVectorField& Cf = mesh_.Cf();
    const surfaceVectorField& Sf = mesh_.Sf();

    const vector Omega = omega_->value(mesh_.time().timeOutputValue())*axis_;

    // Included patches
    forAll(includedFaces_, patchi)
    {
        forAll(includedFaces_[patchi], i)
        {
            label patchFacei = includedFaces_[patchi][i];

            phi[patchi][patchFacei] = 0.0;
        }
    }

    // Excluded patches
    forAll(excludedFaces_, patchi)
    {
        forAll(excludedFaces_[patchi], i)
        {
            label patchFacei = excludedFaces_[patchi][i];

            phi[patchi][patchFacei] -=
                rho[patchi][patchFacei]
              * (Omega ^ (Cf.boundaryField()[patchi][patchFacei] - origin_))
              & Sf.boundaryField()[patchi][patchFacei];
        }
    }
}
```

Relative face fluxes on rotating patches are zero

$$\boldsymbol{u}_R \cdot \boldsymbol{S}_f = 0$$

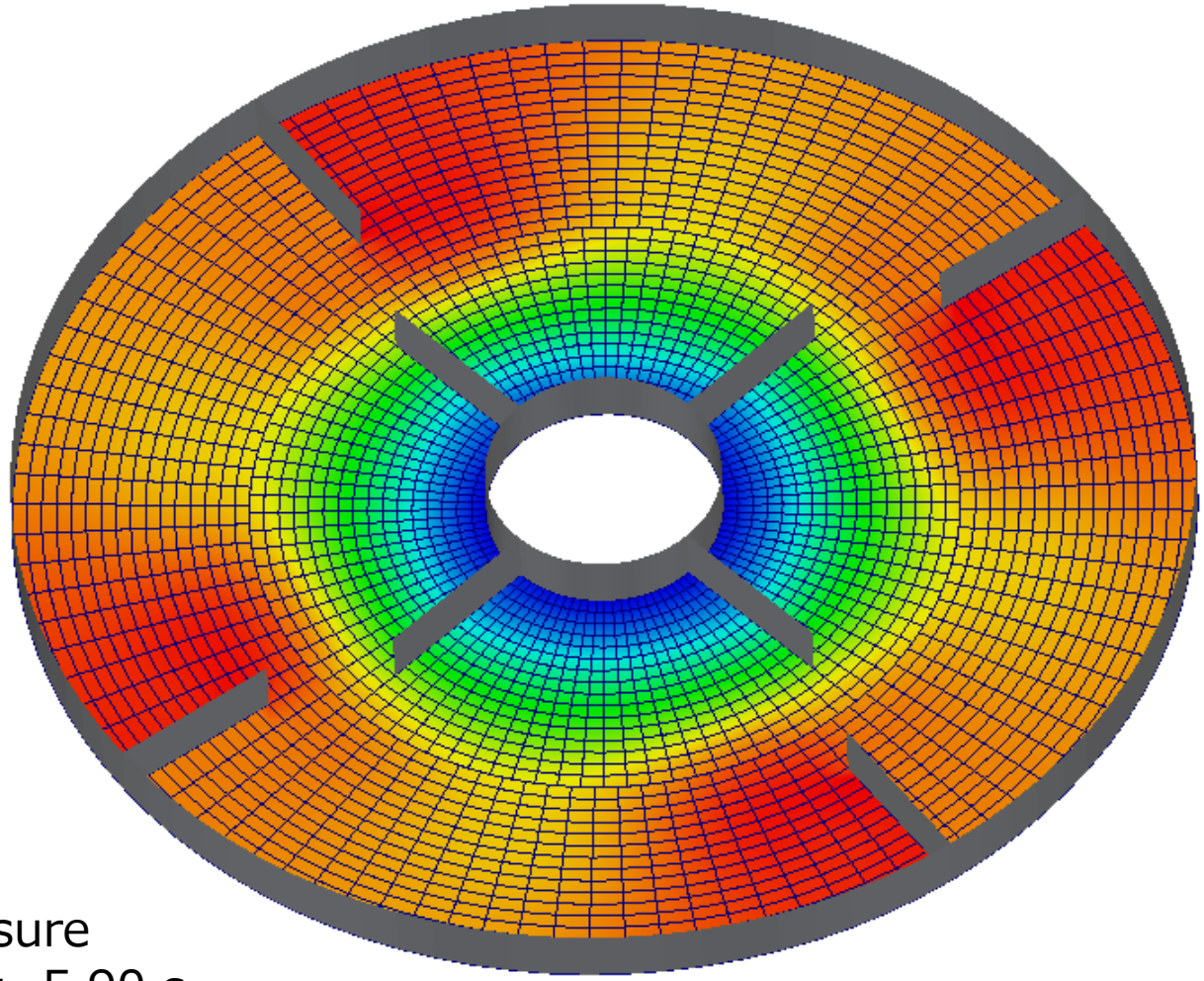
Relative face fluxes on stationary patches in the rotating cell zone is calculated by subtracting $(\boldsymbol{\Omega} \times \boldsymbol{r}) \cdot \boldsymbol{S}_f$ from the absolute face fluxes

Chapter 4 Dynamic Mesh

In this chapter we shall describe how to set up
a transient simulation with mesh motion in OpenFOAM.

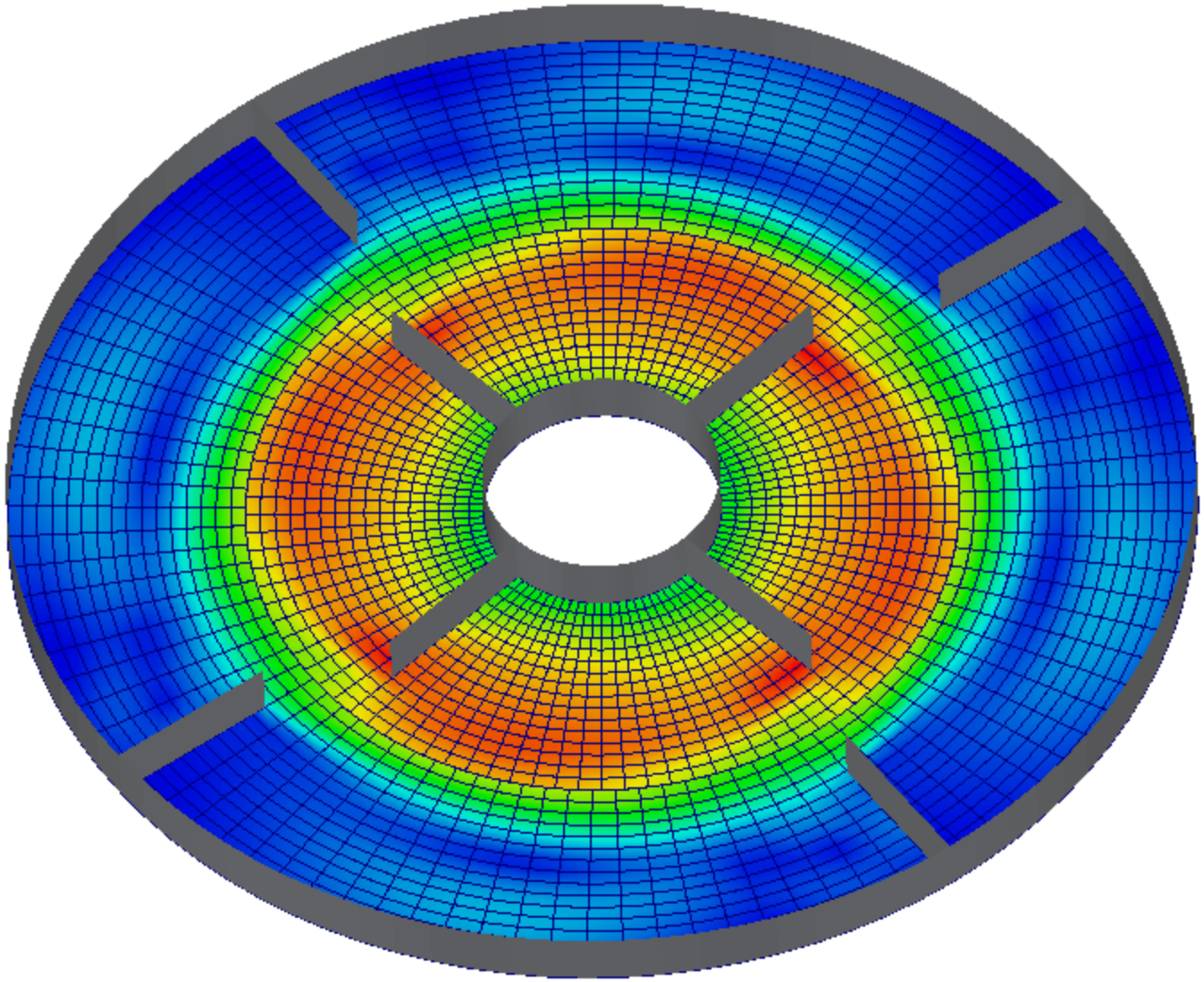
First run the tutorial and enjoy post-processing:

incompressible/pimpleDyMFoam/mixerVesselAMI2D

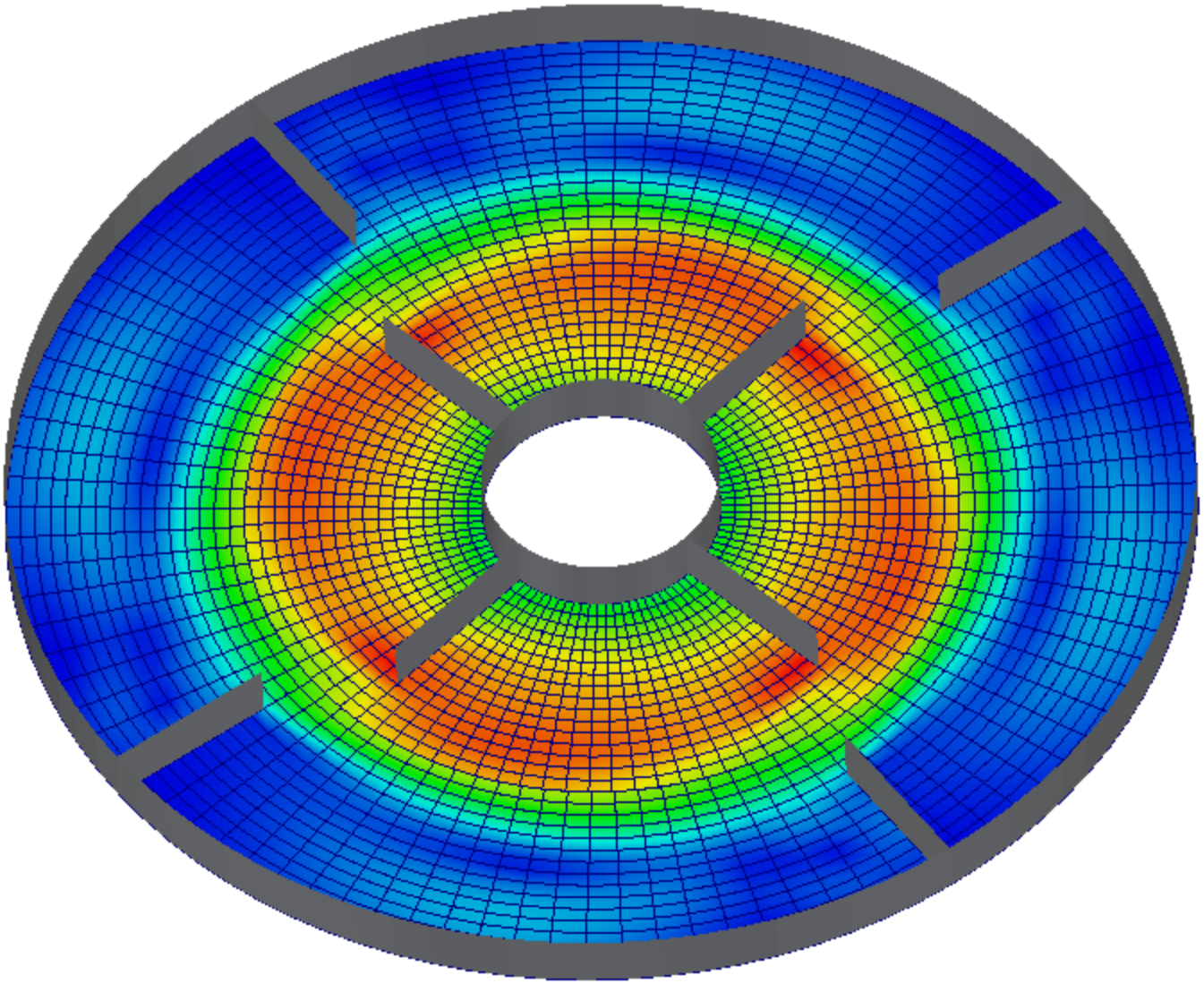


Static Pressure

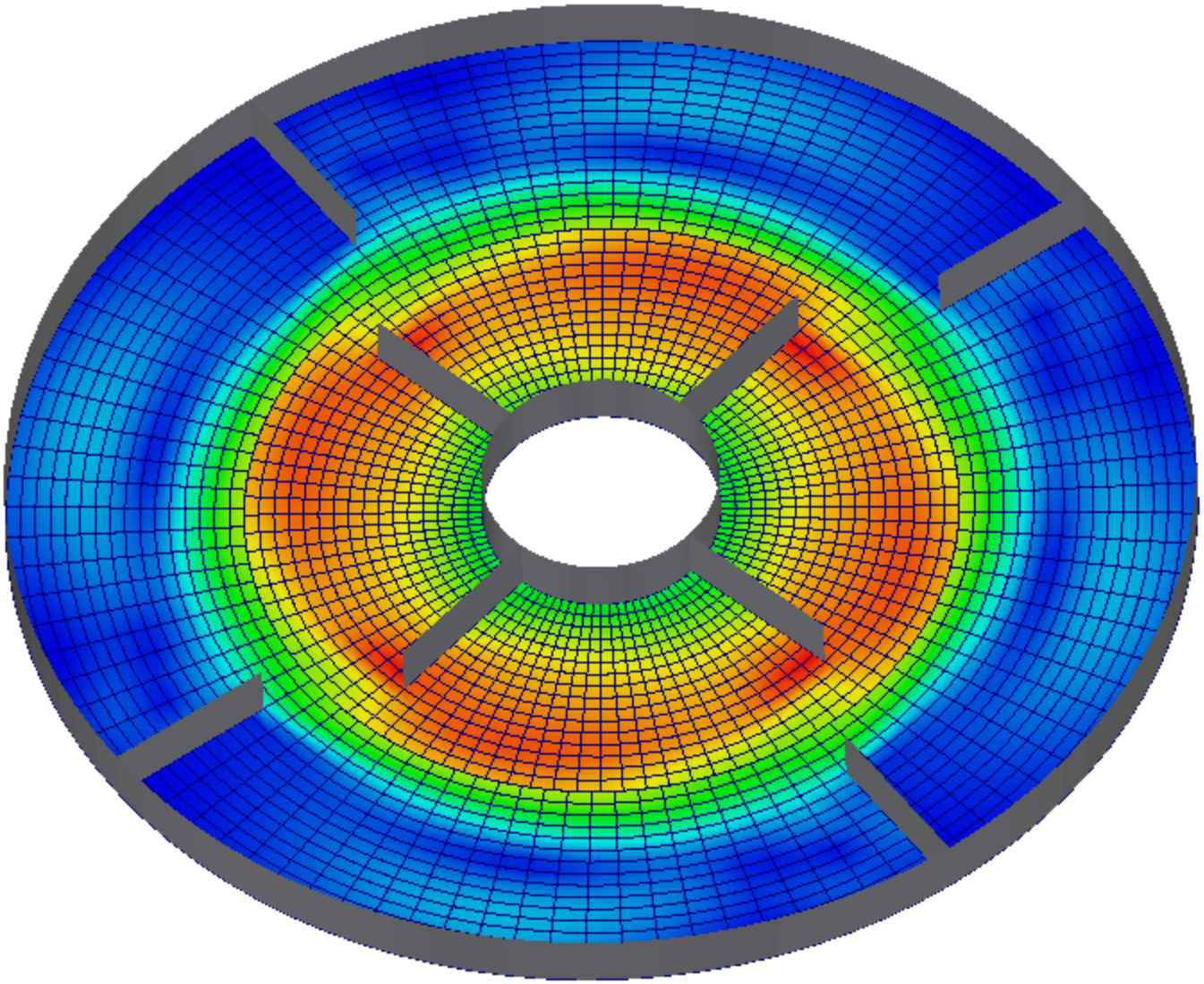
t=5.90 s



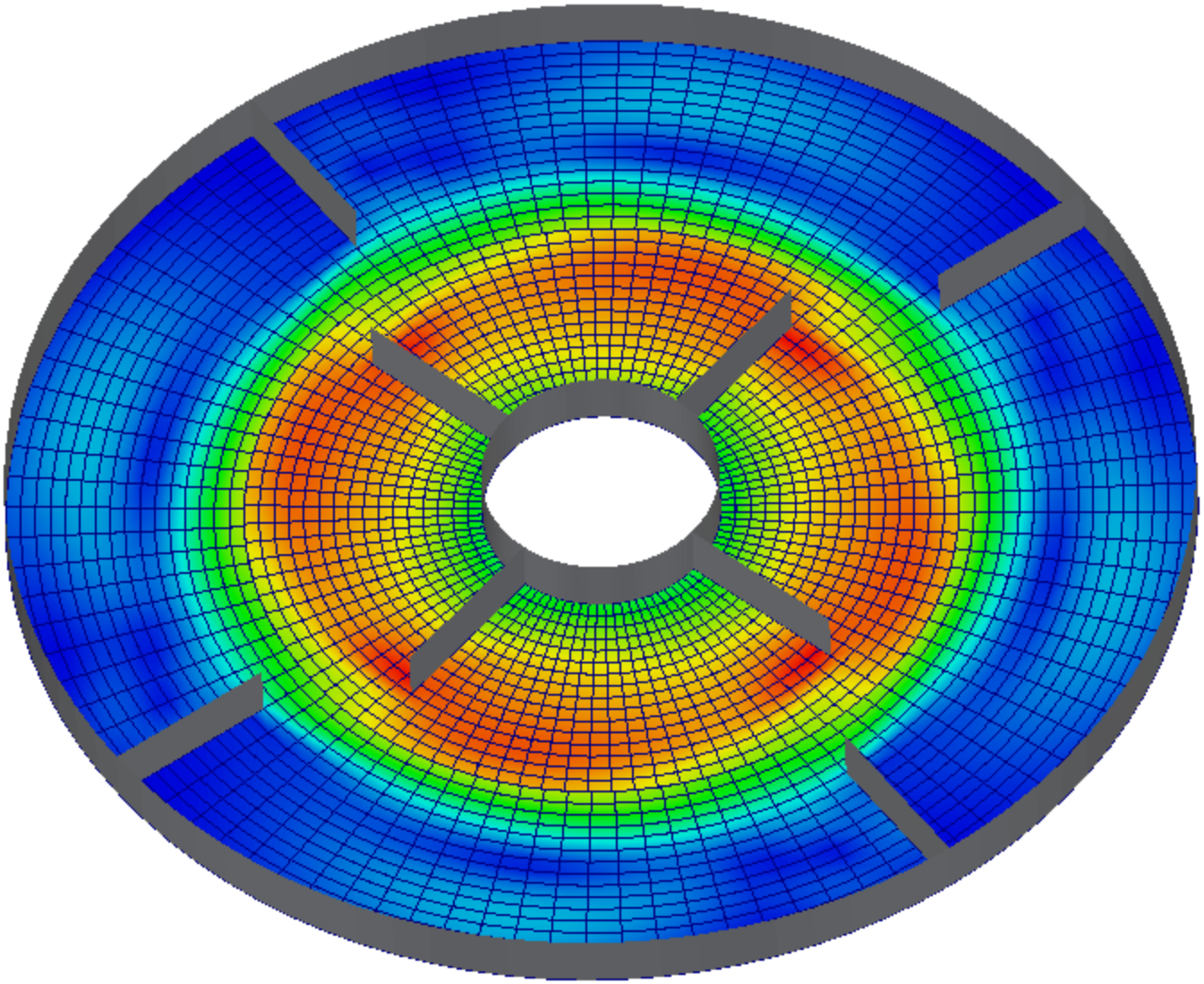
Velocity Magnitude
t=5.90 s



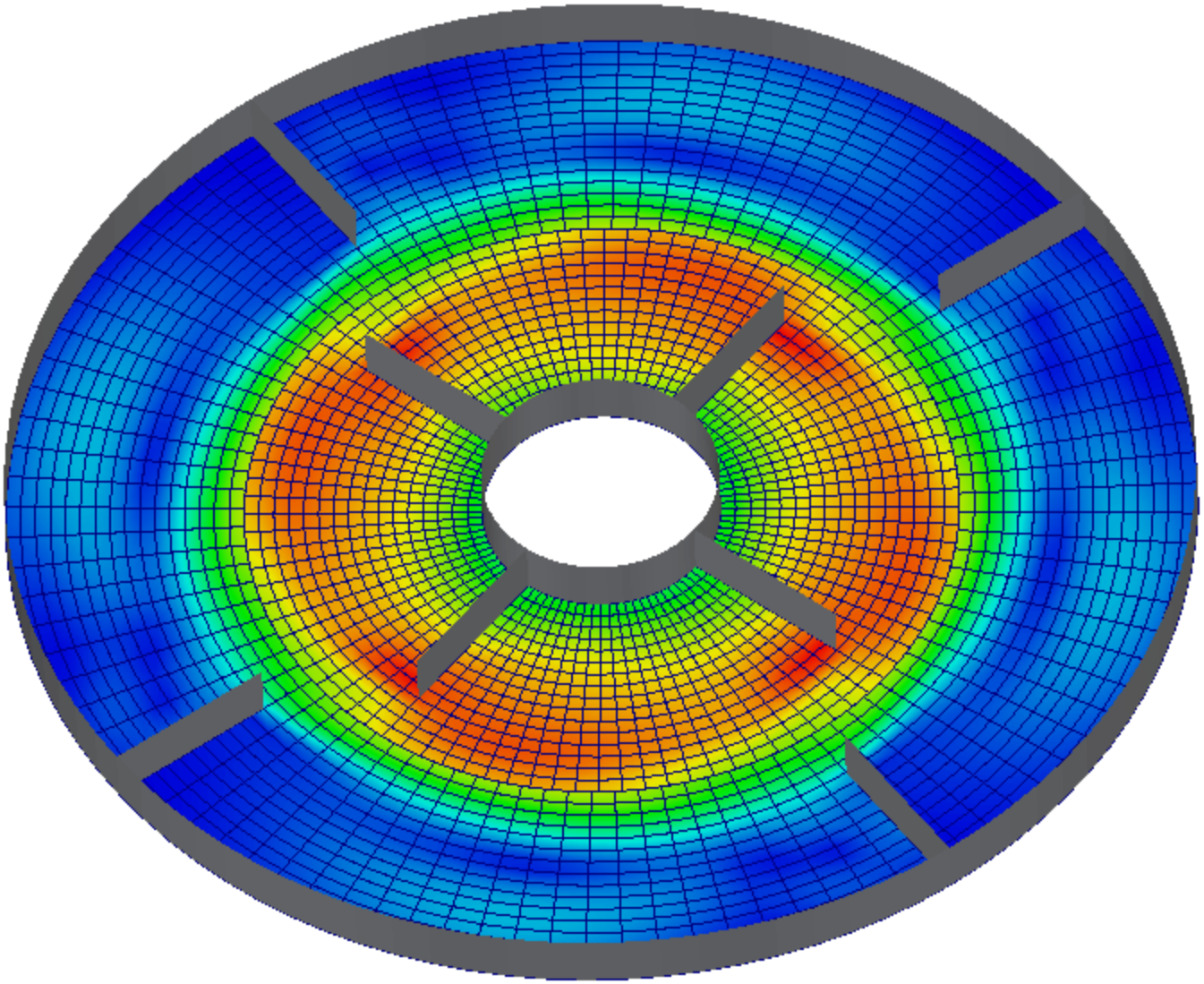
Velocity Magnitude
t=5.905 s



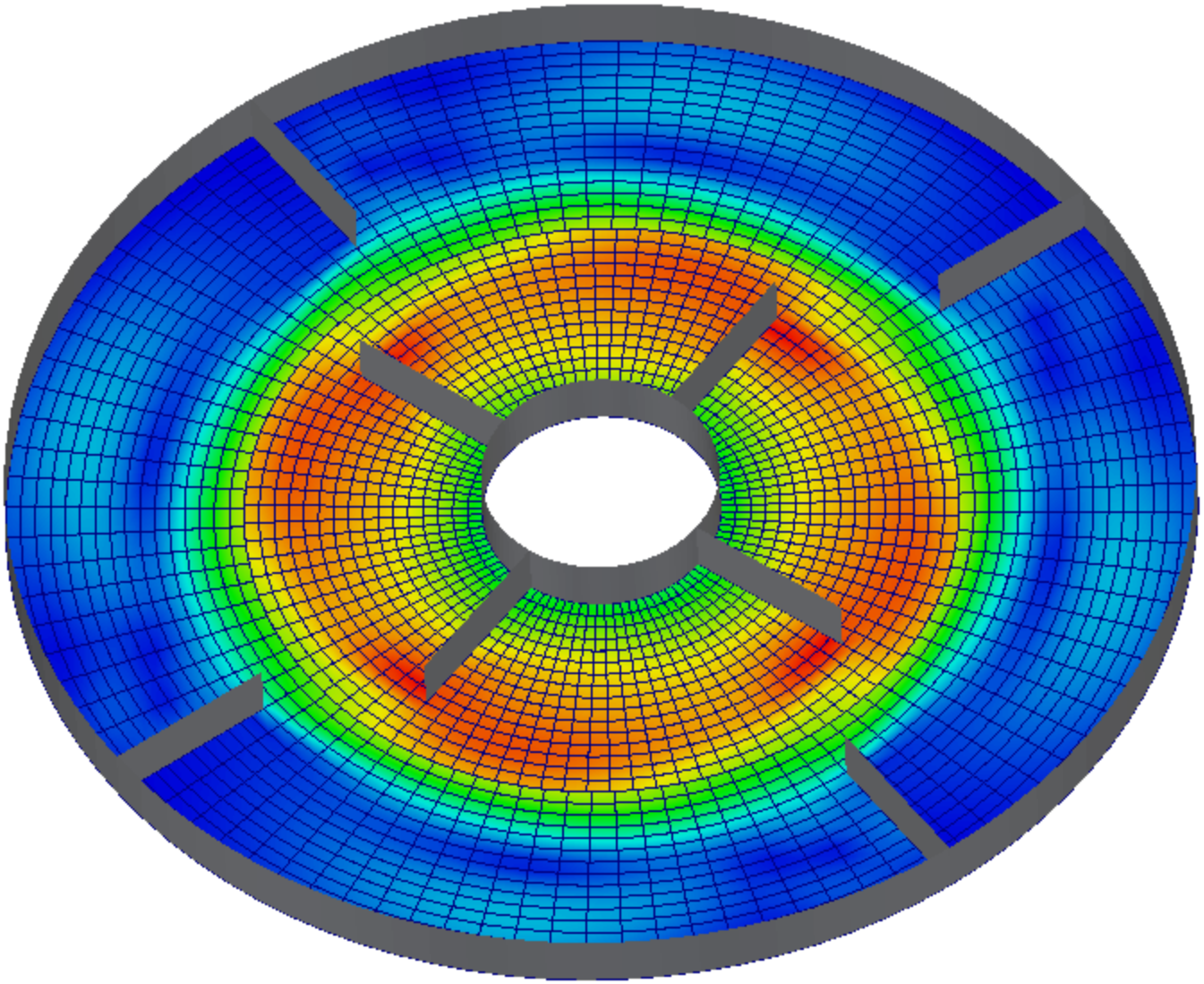
Velocity Magnitude
t=5.91 s



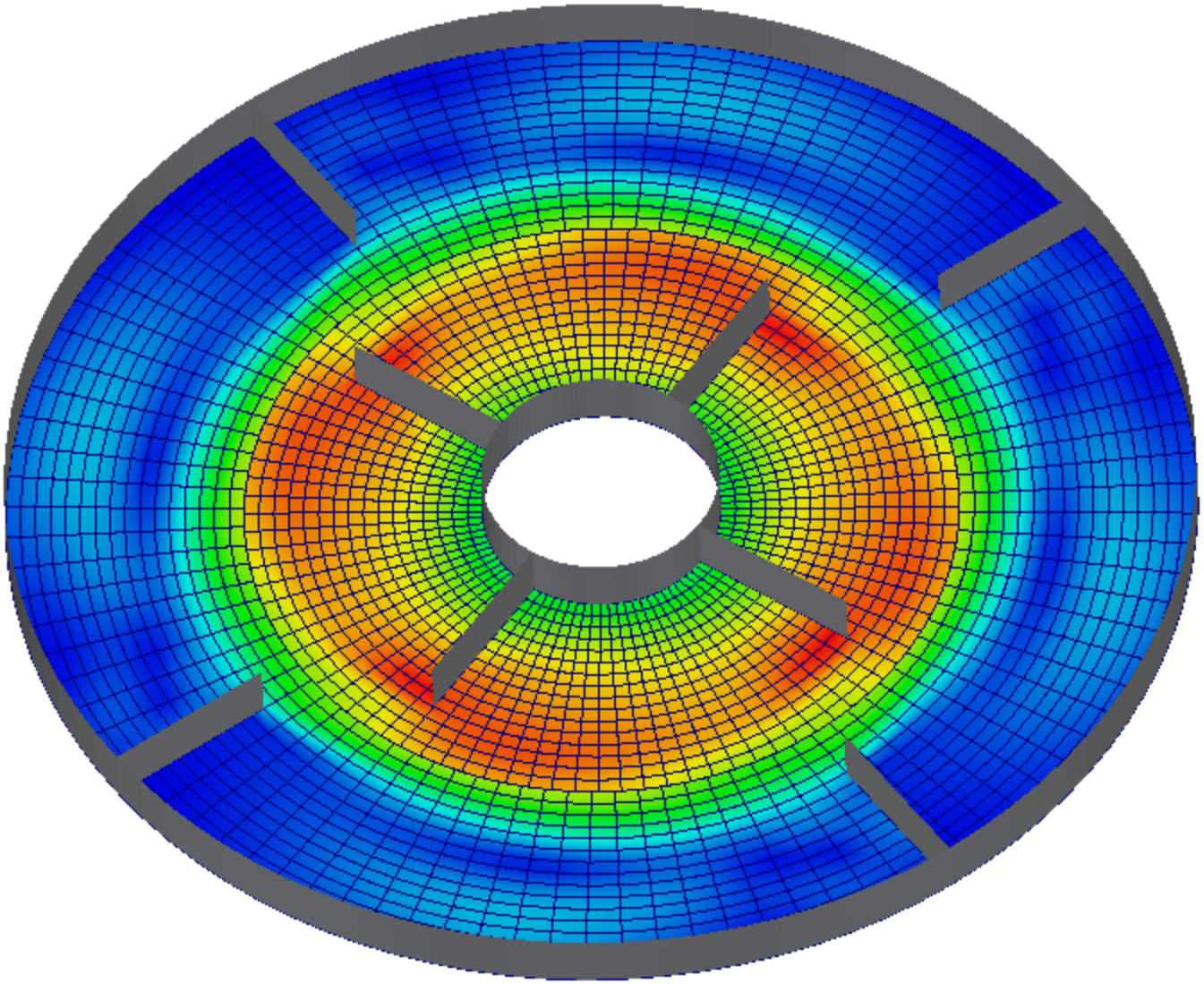
Velocity Magnitude
t=5.915 s



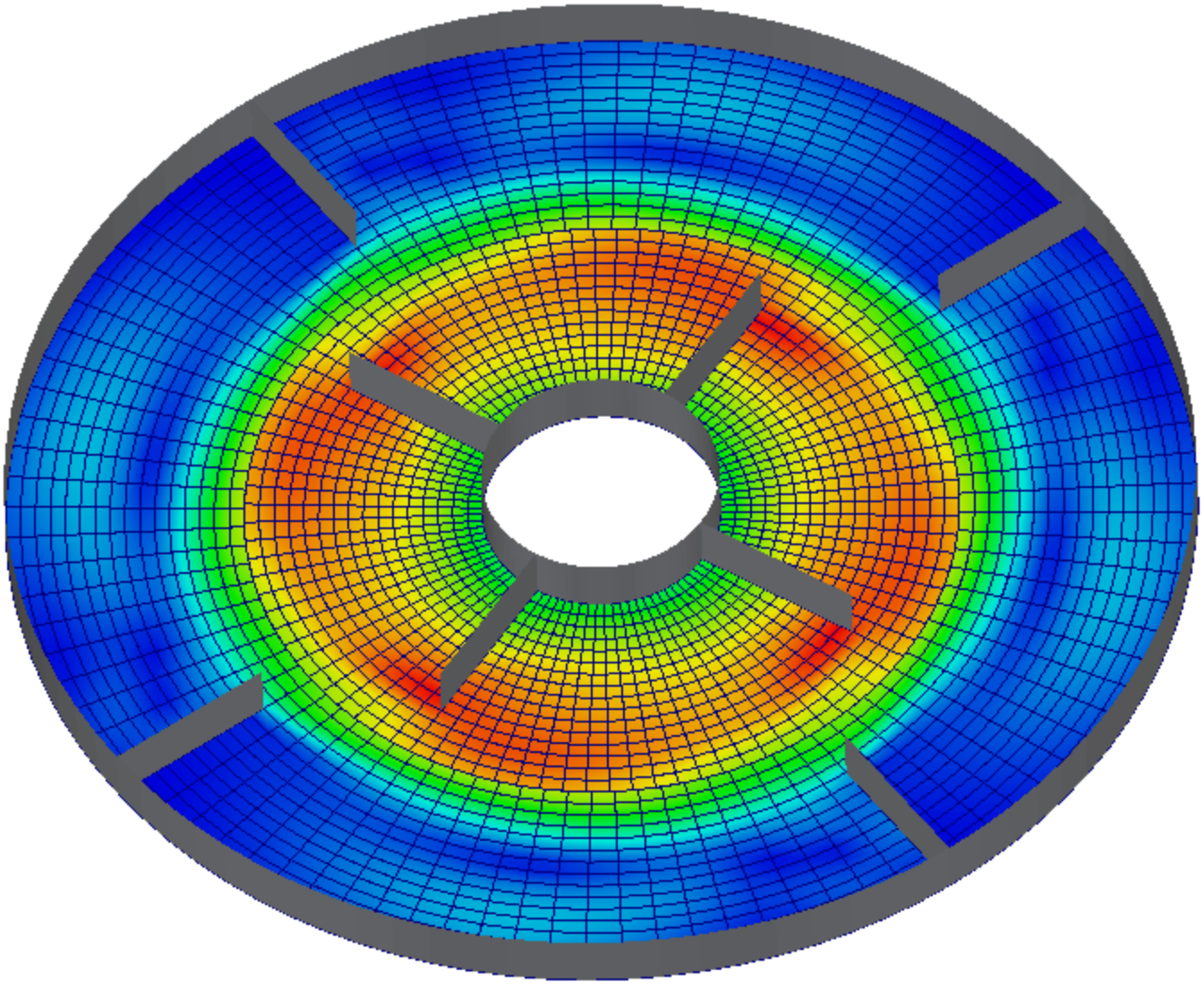
Velocity Magnitude
t=5.92 s



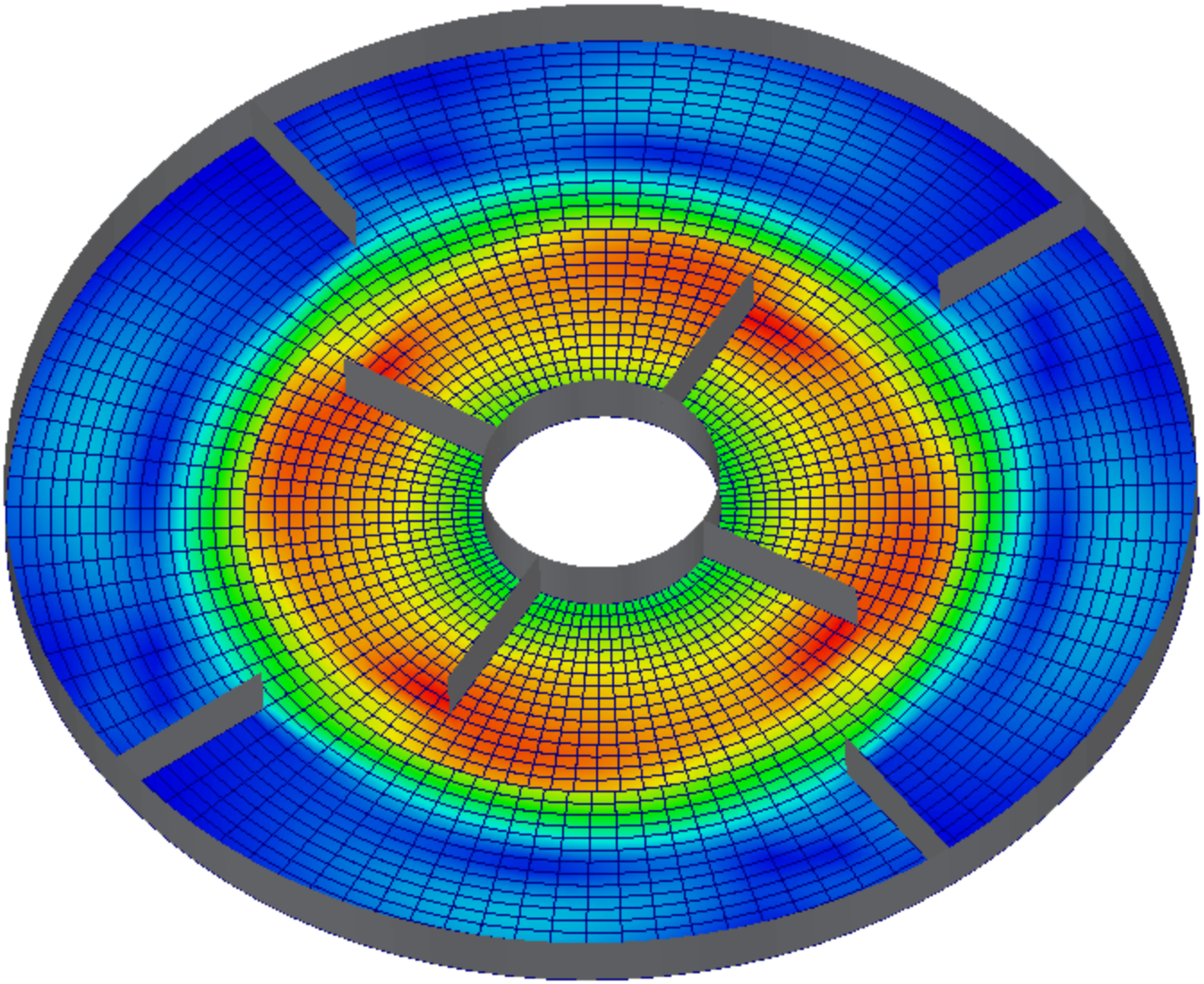
Velocity Magnitude
t=5.925 s



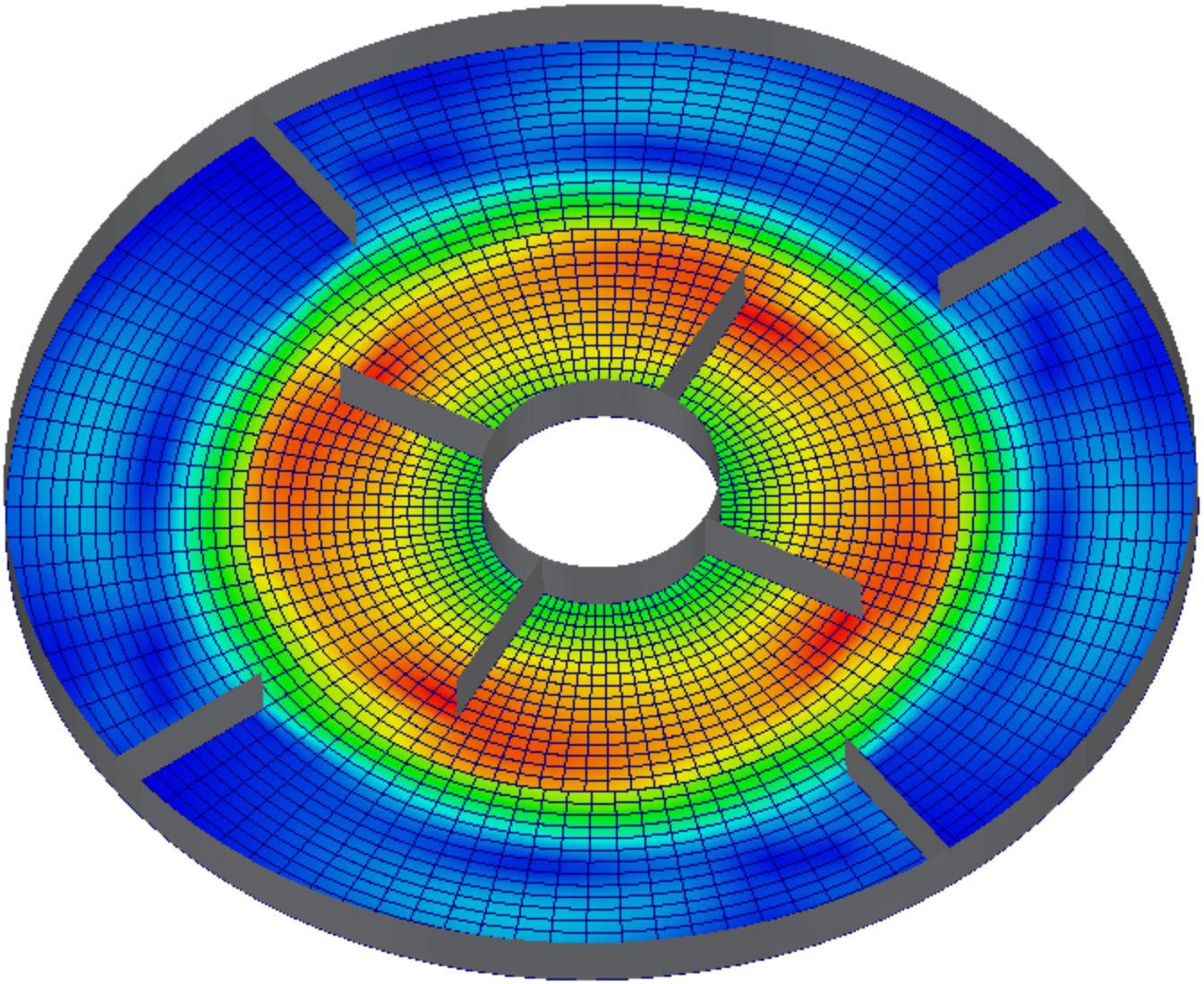
Velocity Magnitude
t=5.93 s



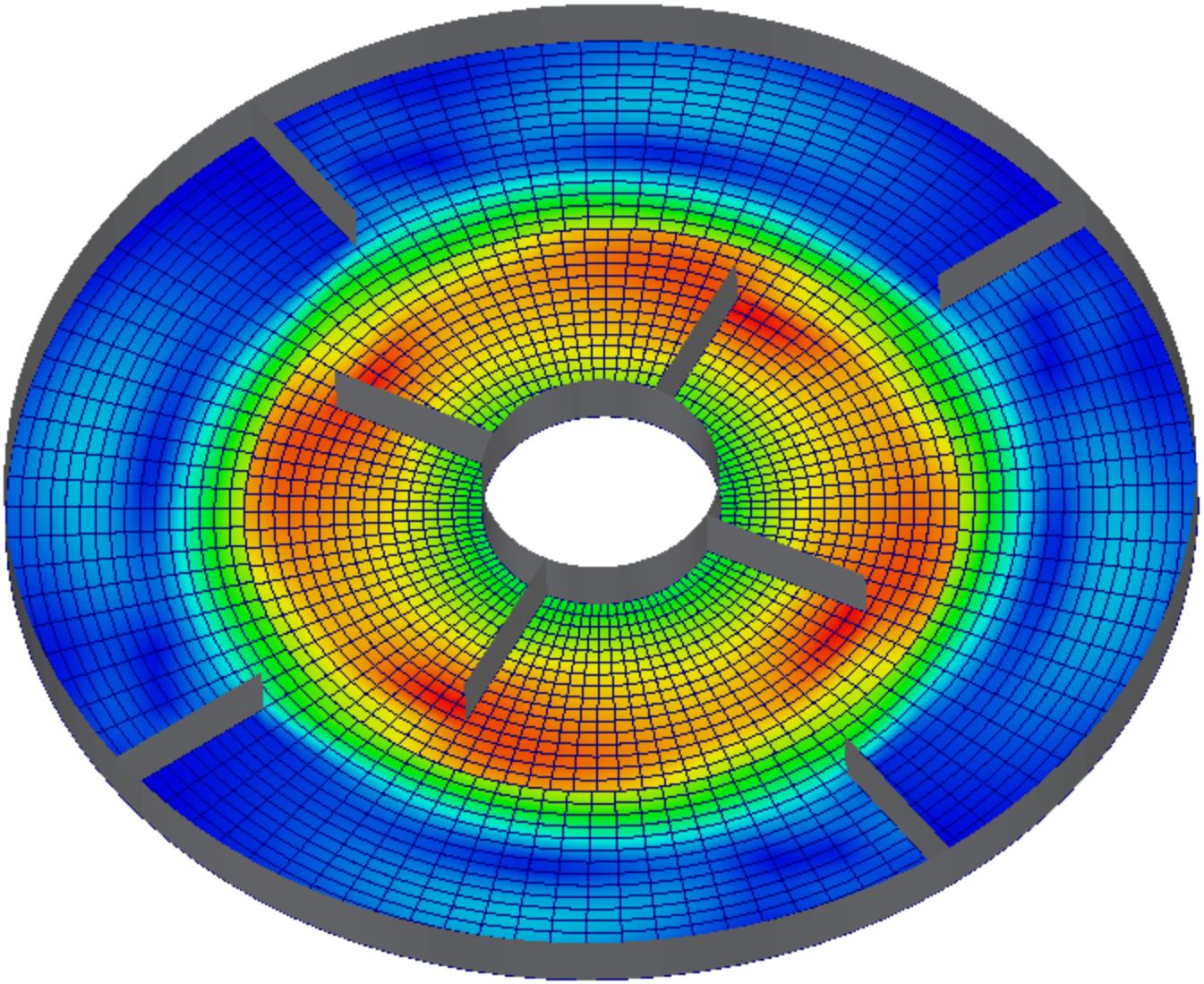
Velocity Magnitude
t=5.935 s



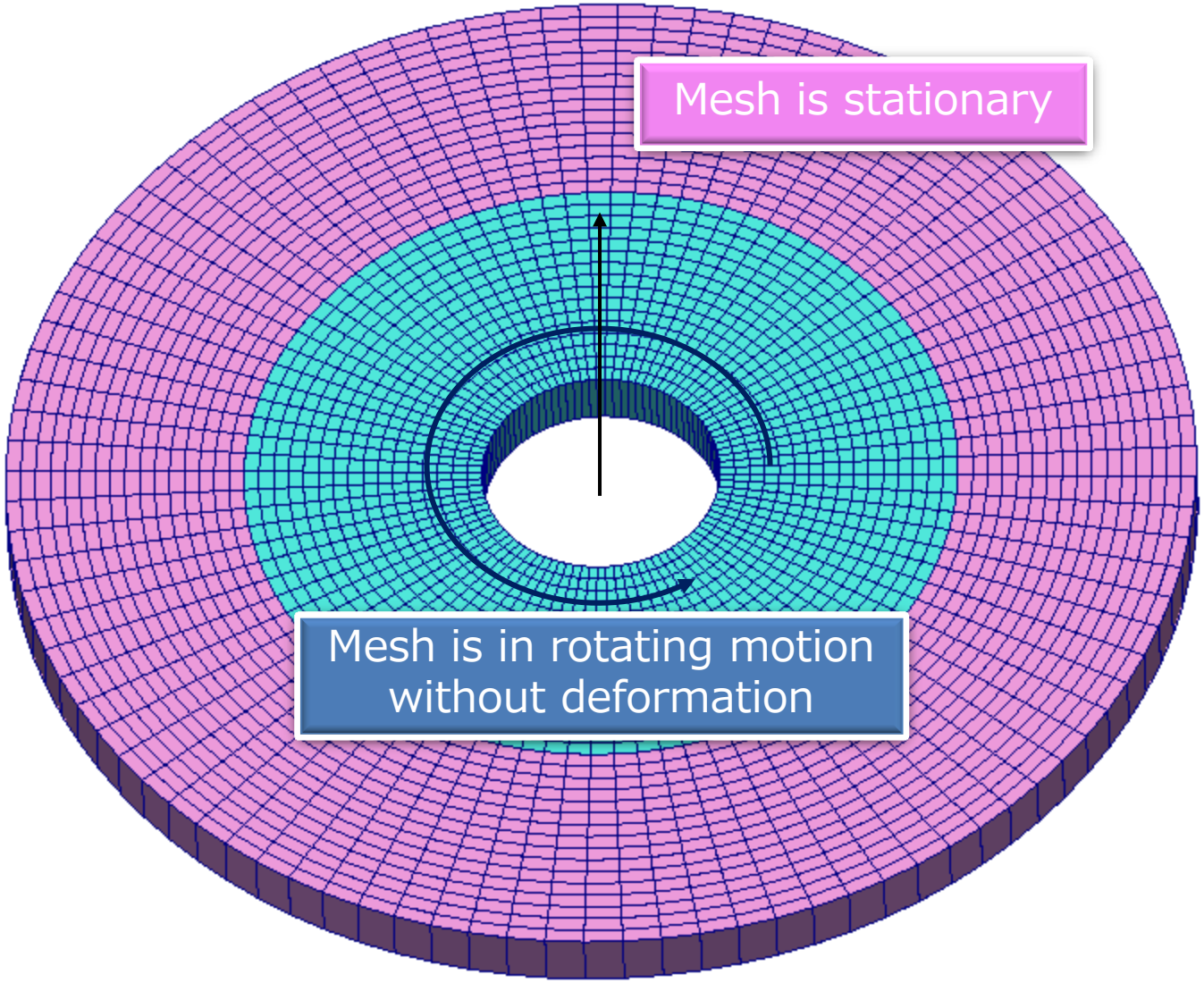
Velocity Magnitude
t=5.94 s



Velocity Magnitude
t=5.945 s



Velocity Magnitude
t=5.95 s

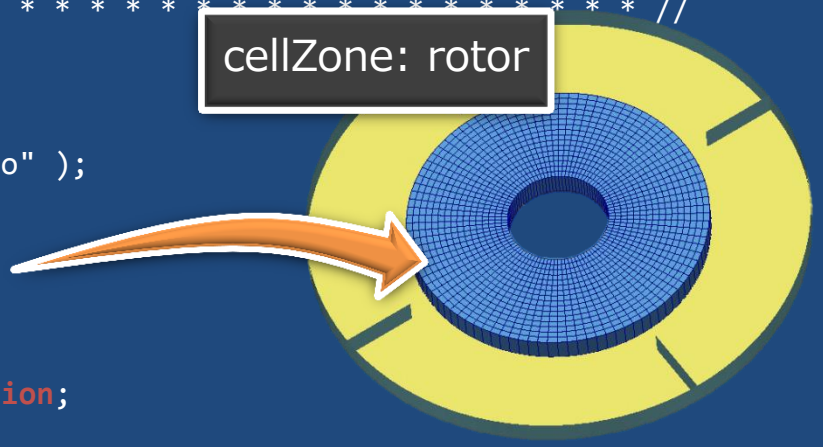


Settings are in the next page.

- Settings of dynamic mesh are specified in ***dynamicMeshDict*** file

constant/dynamicMeshDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh   solidBodyMotionFvMesh;

motionSolverLibs ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
{
    cellZone        rotor;

    solidBodyMotionFunction  rotatingMotion;
    rotatingMotionCoeffs
    {
        origin        (0 0 0);
        axis          (0 0 1);
        omega         6.2832; // rad/s
    }
}
```



The cellZone “rotor” is rotating counterclockwise at 6.2832 rad/s about the origin.

solidBodyMotionFvMesh deals with mesh motions without deformations such as translational and rotational motions.

- The following motions (***solidBodyMotionFunction***) are available for ***solidBodyMotionFvMesh***

Translational motion

- linearMotion
- oscillatingLinearMotion

Rotational motion

- rotatingMotion
- axisRotationMotion
- oscillatingRotatingMotion

Ship Design Analysis

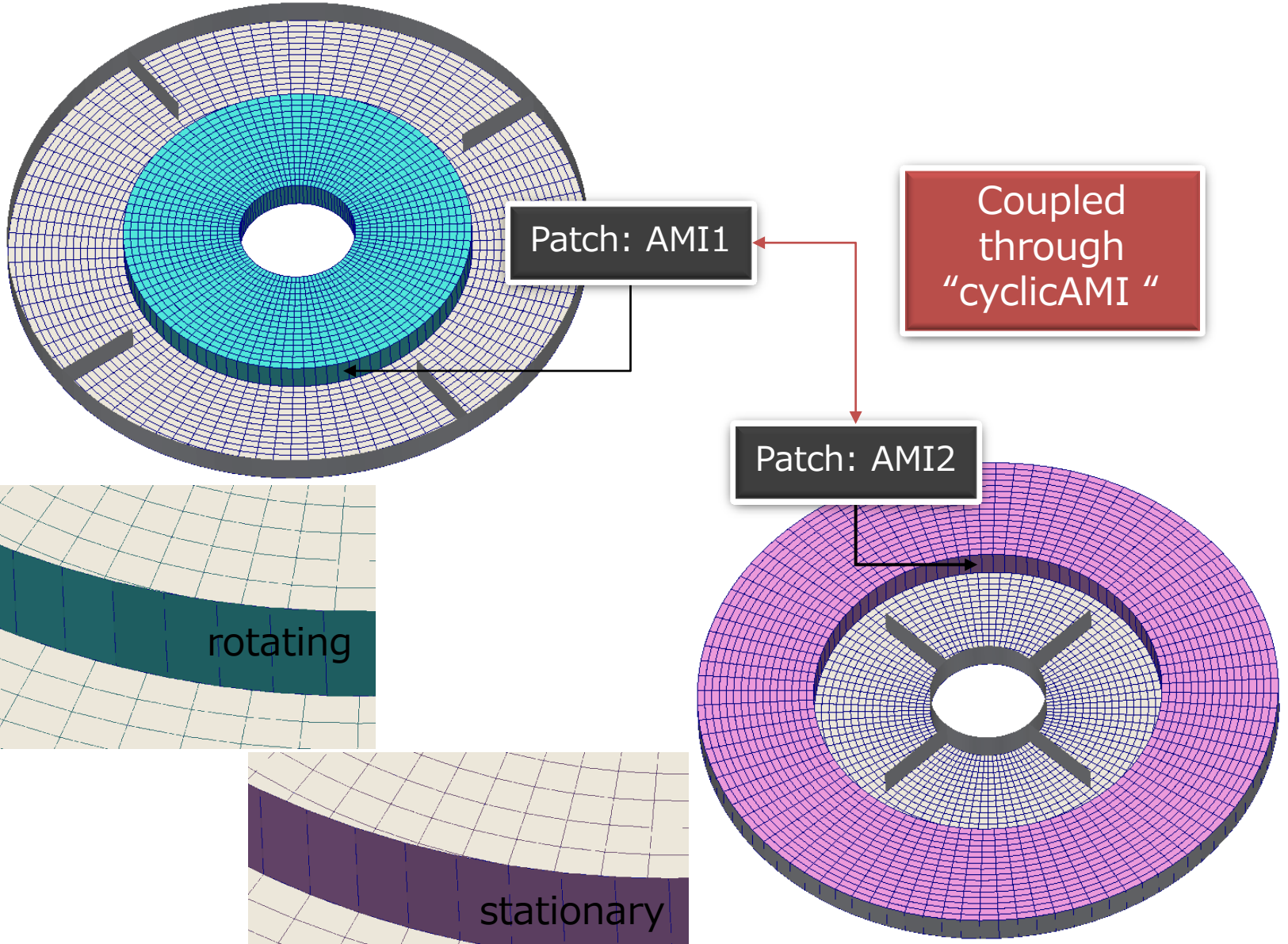
- SDA [4]

Combination of the above motions

- multiMotion

- Code: `src/dynamicFvMesh/solidBodyMotionFvMesh/solidBodyMotionFunctions`

Sliding interface capability using cyclicAMI

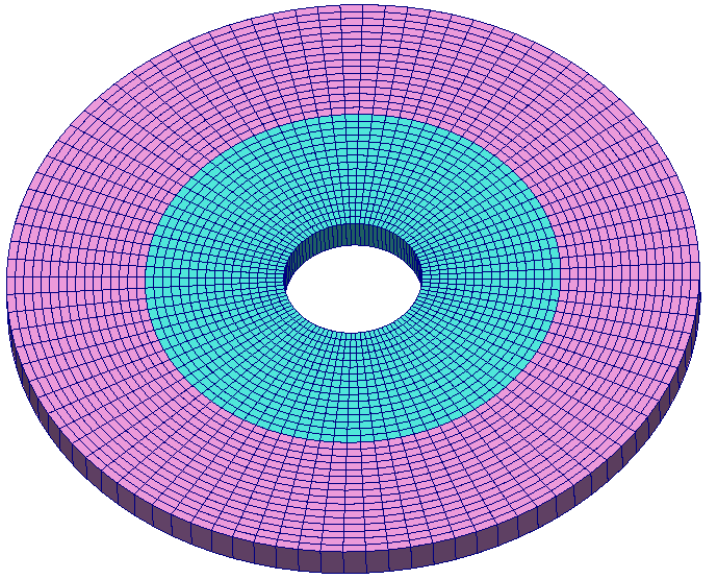


Sliding interface capability using cyclicAMI

constant/polyMesh/boundary

```
    nFaces          192;
    startFace       6048;
}
AMI1
{
    type            cyclicAMI;
    inGroups        1(cyclicAMI);
    nFaces          96;
    startFace       6240;
    matchTolerance  0.0001;
    transform       noOrdering;
    neighbourPatch  AMI2;
}
AMI2
{
    type            cyclicAMI;
    inGroups        1(cyclicAMI);
    nFaces          96;
    startFace       6336;
    matchTolerance  0.0001;
    transform       noOrdering;
    neighbourPatch  AMI1;
}
front
{
    type            empty;
    inGroups        1(empty);
```

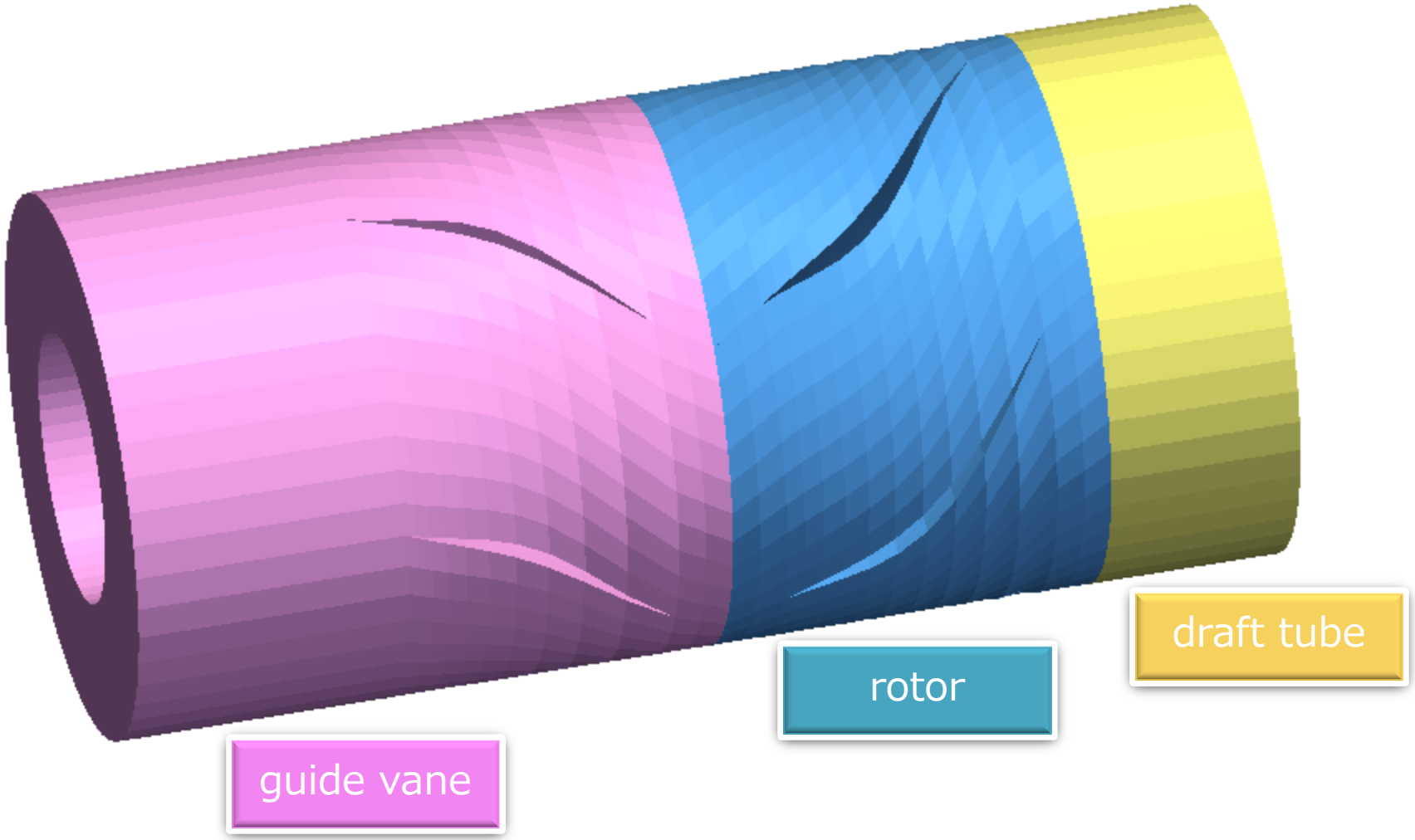
- Sliding interface capability , *e.g.* for rotating machinery is available using **cyclicAMI** boundary patch class.
- In "**neighbourPatch**", the coupled patch name is specified.



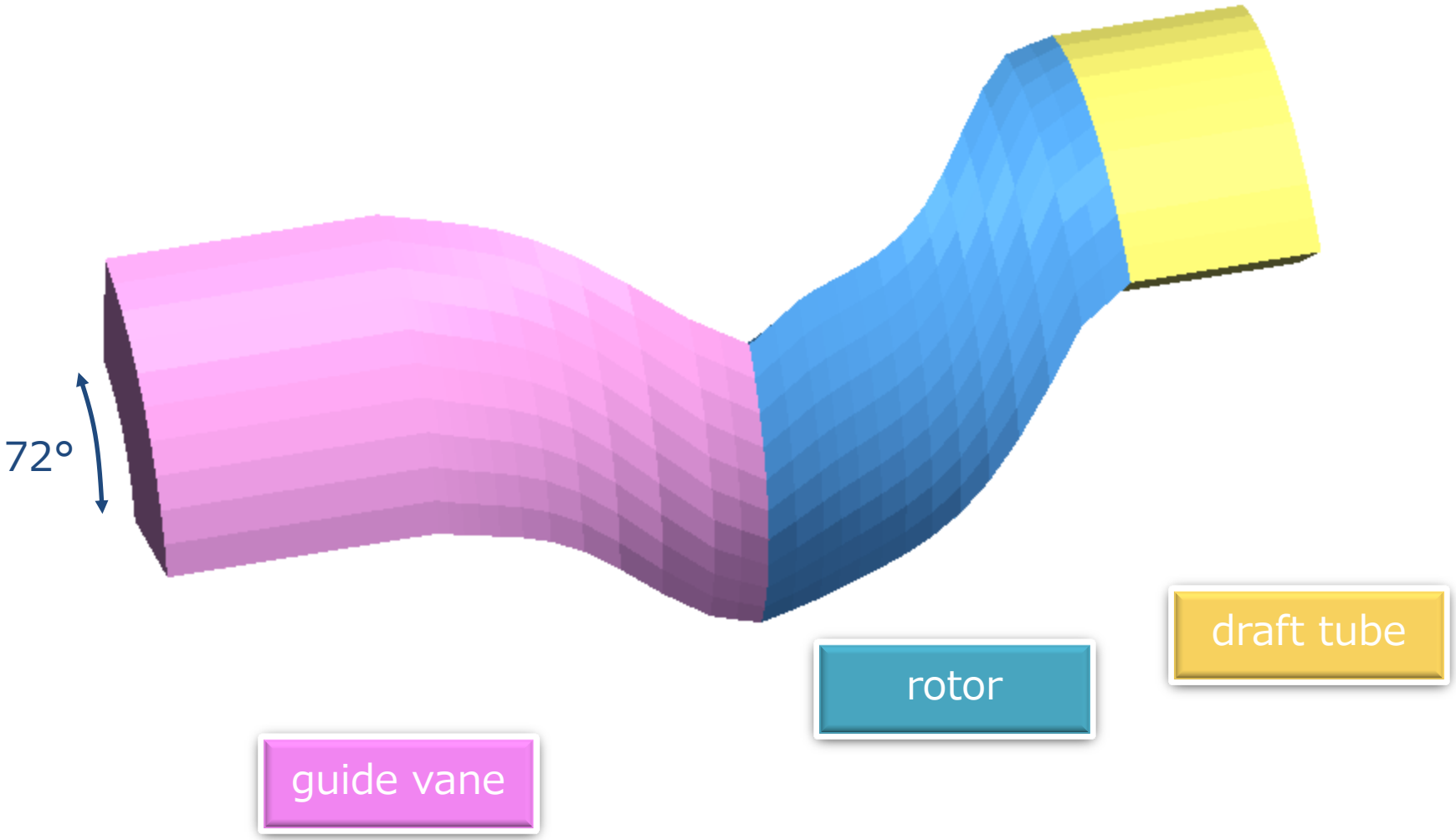
Chapter 5 Mixing plane interface

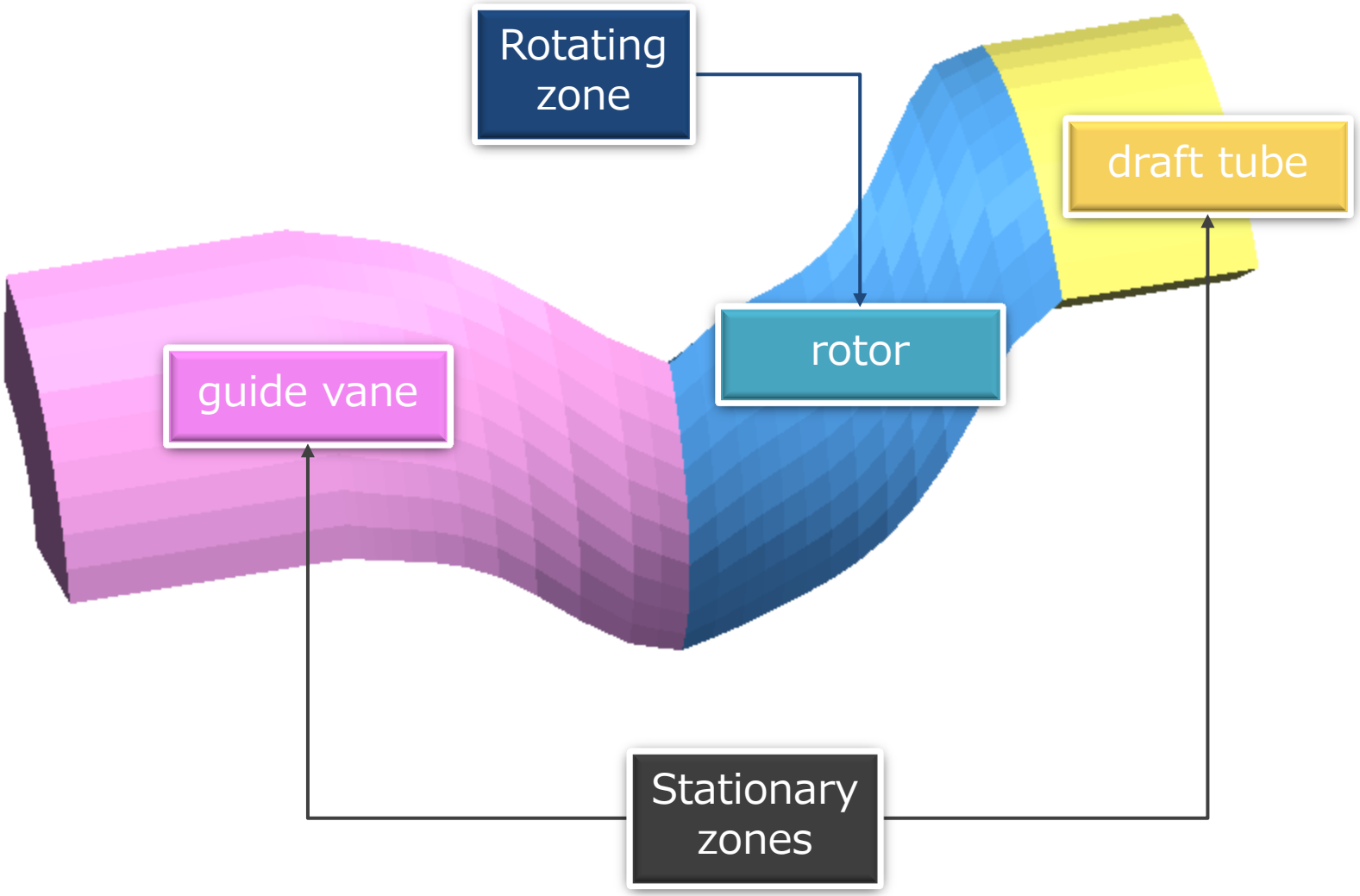
In this chapter we shall describe how to use
the mixing plane interface in OpenFOAM extend version,
the community-driven version.

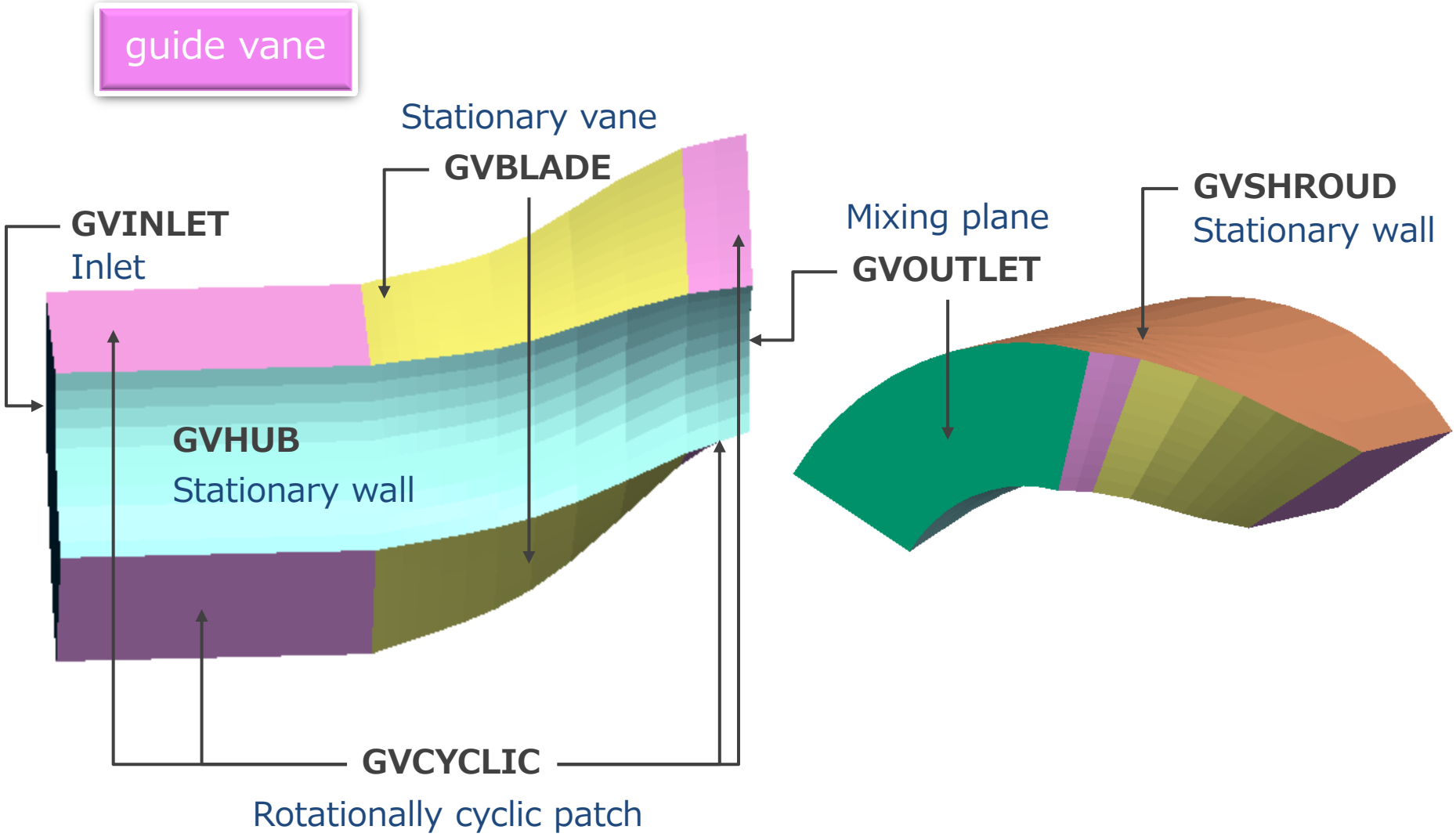
- foam-extend-3.1/tutorials/incompressible/MRFSimpleFoam/axialTurbine_mixingPlane

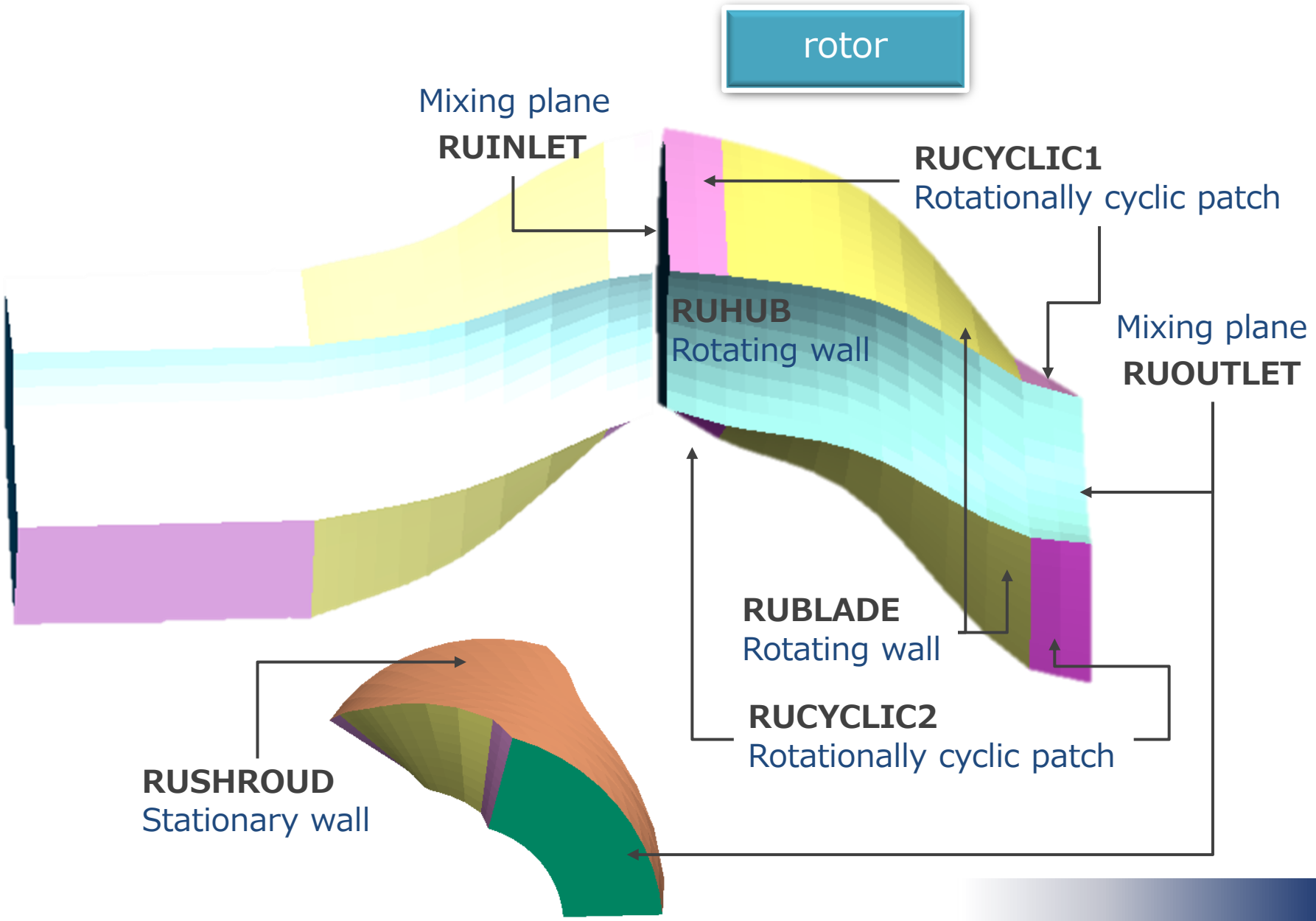


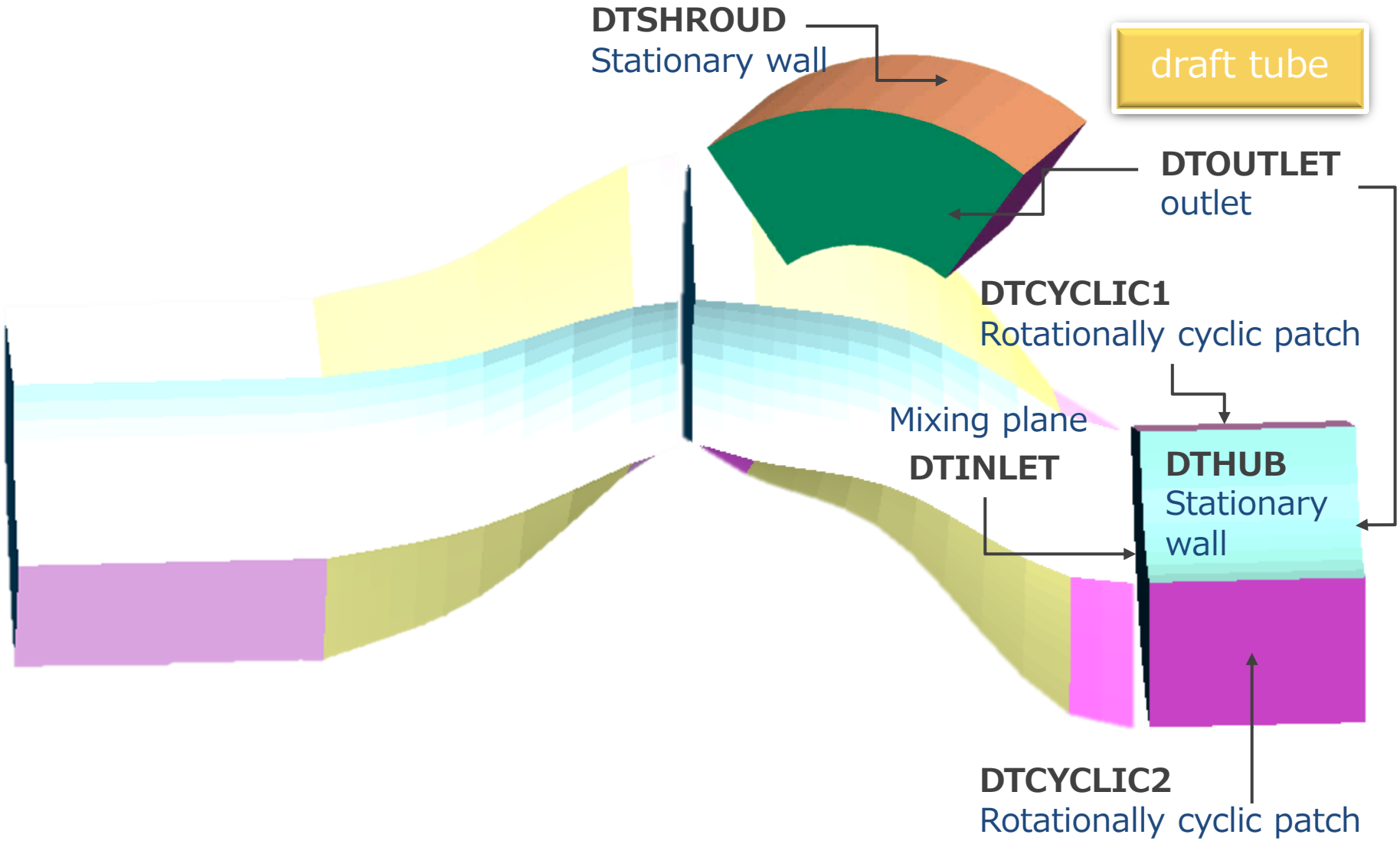
- Only fifth part of the geometry is computed by making use of the rotational periodicity




Rotating zone
draft tube
rotor
guide vane
Stationary zones


guide vane
Stationary vane
GVBLADE
GVINLET
Inlet
Mixing plane
GVOUTLET
GVSHROUD
Stationary wall
GVHUB
Stationary wall
GVCYCLIC
Rotationally cyclic patch


rotor
Mixing plane
RUINLET
RUCYCLIC1
Rotationally cyclic patch
RUHUB
Rotating wall
Mixing plane
RUOUTLET
RUBLADE
Rotating wall
RUCYCLIC2
Rotationally cyclic patch
RUSHROUD
Stationary wall


DTSHROUD
Stationary wall
draft tube
DTOUTLET
outlet
DTCYCLIC1
Rotationally cyclic patch
Mixing plane
DTINLET
DTHUB
Stationary wall
DTCYCLIC2
Rotationally cyclic patch

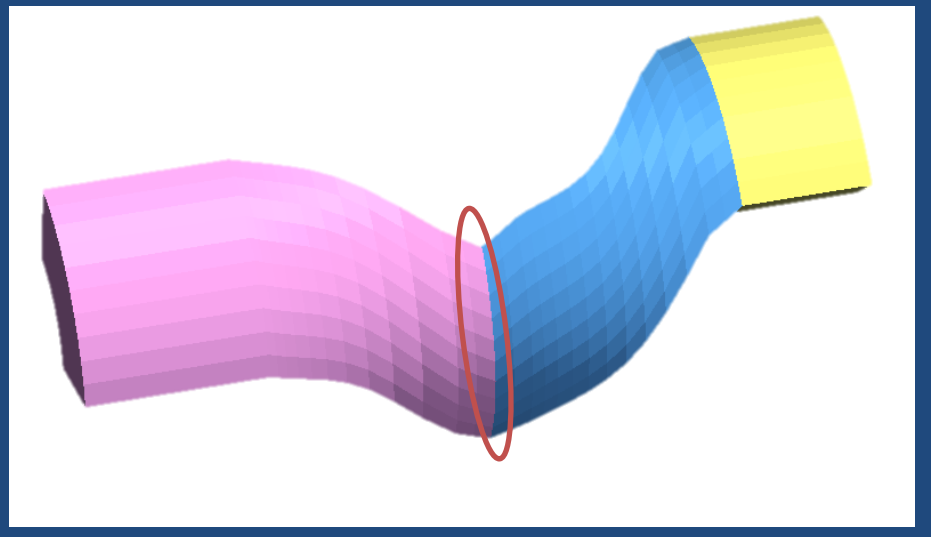
- Specifying the mixing plane in ***boundary*** file

constant/polyMesh/boundary

```
GVOUTLET
{
    type            mixingPlane;
    nFaces          100;
    startFace       11840;
    shadowPatch     RUINLET;
    zone            GVOUTLETZone;
    coordinateSystem
    {
        type            cylindrical;
        name            mixingCS;
        origin          (0 0 0);
        e1              (1 0 0);
        e3              (0 0 1);
        inDegrees       true;
    }
    ribbonPatch
    {
        sweepAxis       Theta;
        stackAxis       R;
        discretisation  bothPatches;
    }
}
```

```
RUINLET
{
    type            mixingPlane;
    nFaces          100;
    startFace       12820;
    shadowPatch     GVOUTLET;
    zone            RUINLETZone;
}
```

Circumferential averaging



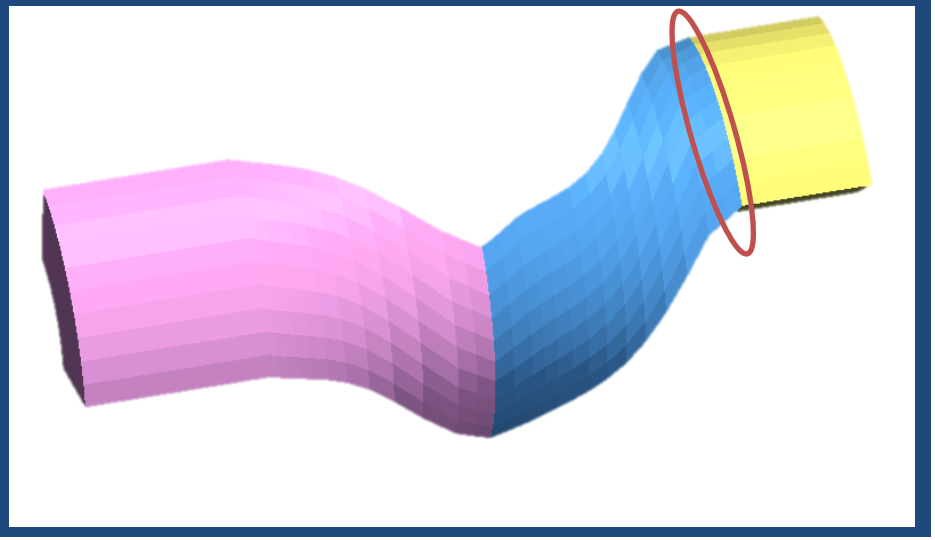
- Specifying the mixing plane in ***boundary*** file

constant/polyMesh/boundary

```
RUOUTLET
{
    type            mixingPlane;
    nFaces          100;
    startFace       12920;
    shadowPatch     DTINLET;
    zone            RUOUTLETZone;
    coordinateSystem
    {
        type            cylindrical;
        name            mixingCS;
        origin          (0 0 0);
        e1              (1 0 0);
        e3              (0 0 1);
        inDegrees       true;
    }
    ribbonPatch
    {
        sweepAxis       Theta;
        stackAxis       R;
        discretisation  bothPatches;
    }
}
```

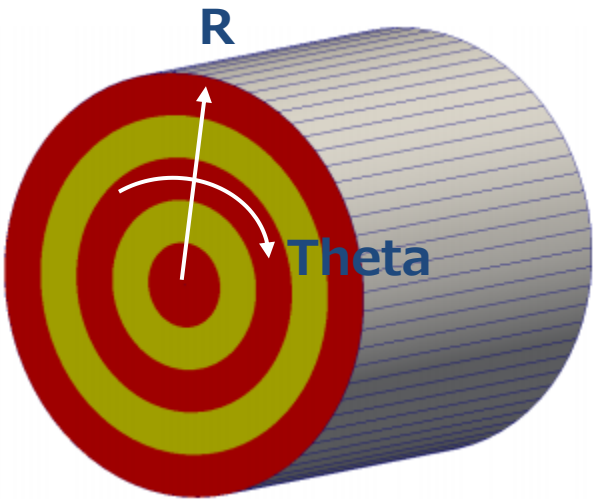
```
DTINLET
{
    type            mixingPlane;
    nFaces          100;
    startFace       13580;
    shadowPatch     RUOUTLET;
    zone            DTINLETZone;
}
```

Circumferential averaging



- What is the ***ribbonPatch***? [5]

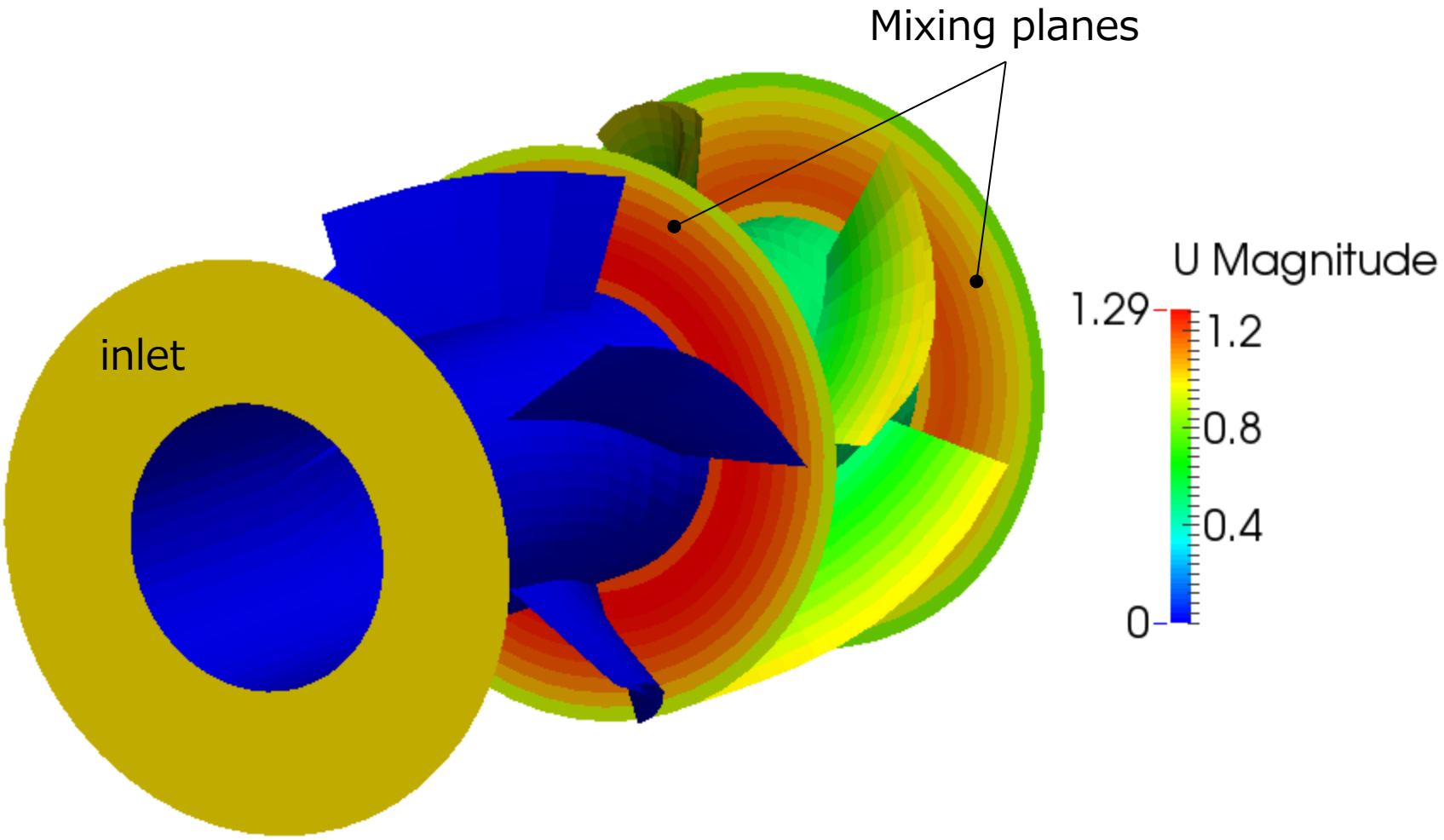
```
ribbonPatch
{
    sweepAxis       Theta;
    stackAxis       R;
    discretisation  bothPatches;
}
```



stackAxis = R

sweepAxis = Theta

- According to [6]
  the ribbon patch will specify the direction in which it will perform the average (***sweepAxis***) and in which direction it will stack the cells (***stackAxis***).


Mixing planes
inlet
U Magnitude
1.29
1.2
0.8
0.4
0

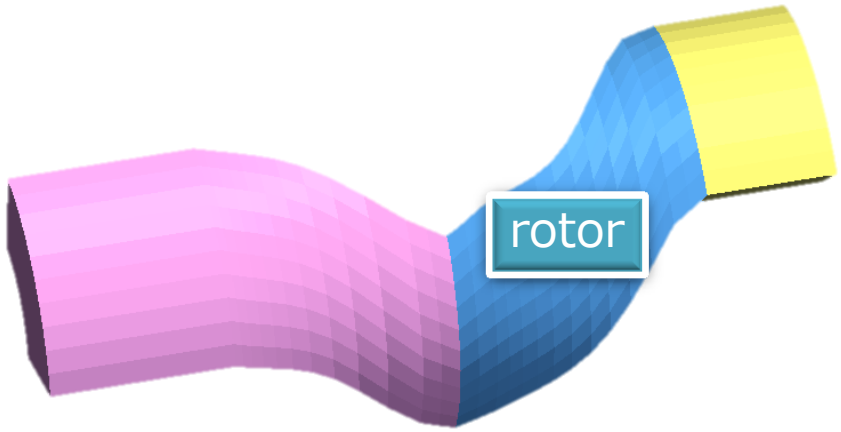
- Specifying the rotating zone in ***MRFZones*** file

constant/MRFZones

```
1
(
    rotor
    {
        // Fixed patches (by default they 'move' with the MRF zone)
        nonRotatingPatches ( RUSHROUD );

        origin    origin [0 1 0 0 0 0 0]  (0 0 0);
        axis      axis   [0 0 0 0 0 0 0]  (0 0 1);
        omega     omega  [0 0 -1 0 0 0 0] -10;
    }
)
```

Name of rotating cellZones

The absolute velocity on the "***nonRotatingPatches***" is **0**.



- [7]

4.1. The RC1 version

The first implementation of the mixingPlane, dubbed the *RC1* version, laid out the basic infrastructure necessary for this kind of coupling interface [12]. This initial design is reusing the generalized grid interface (GGI) as the underlying area-weighted interpolator and introduced a novel mesh-driven discretization algorithm for the determination of the circumferential interpolation bands. This first basic mixingPlane interpolation scheme is called *areaAveraging*. Still, extensive validation tests of this initial design have shown the limitations of an area-weighted-only interpolator for the circumferential averaging of flow fields. In the presence of large flow gradients very close to the interface, it was shown that the mixingPlane interface was still mass-conservative, but at the expense of disturbances of the flow fields in the vicinity of the interface, mostly for the pressure field [13]. The behaviour of the mixingPlane was demonstrated using a synthetic 2D test case named domAdomB [14], in which two stationary flow passages are artificially coupled by a mixingPlane. The test case was published and documented in details [13] [14] [15] to illustrate the current limitations of the RC1 design, and to help in the validation and improvement of this development.

4.2. The RC2 version

A major improvement to the mixingPlane code came in 2013, shortly before the 8th OpenFOAM Workshop held in Jeju, South Korea, thanks to the help and sound advices of the Turbomachinery Working Group members. For this RC2 version, a mass-flow averaging interpolator was introduced in order to complement the area-weighted interpolator algorithm. Additional mixingPlane interpolation schemes were introduced in this version, namely the *fluxAveraging* and *zeroGradient* mixingPlane schemes. Using various mixingPlane specific schemes or "mixing recipes", it was now possible for example to apply the mass-flow averaging interpolator only to the velocity field, while still using the area-averaging interpolator for the other fields.

Using the same domAdomB 2D test case, we were able to show that the use of the mass-flow interpolator for the velocity field greatly diminishes the disturbances observed close to the mixingPlane interface, but at the expense of a significant mass-flow loss at the interface. Basically, the RC2 version of the mixingPlane was no longer mass conservative when using the mass-flow averaging interpolator for the velocity field.

4.3. The RC3 version

Based on the results obtained with the RC1 and RC2 versions of the mixingPlane interface, a new RC3 version was designed shortly after the Jeju OpenFOAM Workshop. The RC3 version basically combines the mass conservation quality of the RC1 version with the mass-flow averaging algorithm of the RC2 version.

Implementation | Related source code

- src/finiteVolume/fields/fvPatchFields/constraint/mixingPlane
  - mixingPlaneFvPatchField.C
  - mixingPlaneFvPatchField.H
  - mixingPlaneFvPatchFields.C
  - mixingPlaneFvPatchFields.H
  - mixingPlaneFvPatchFieldsFwd.H

- src/finiteVolume/fields/fvsPatchFields/constraint/mixingPlane
  - mixingPlaneFvsPatchField.C
  - mixingPlaneFvsPatchField.H
  - mixingPlaneFvsPatchFields.C
  - mixingPlaneFvsPatchFields.H
  - mixingPlaneFvsPatchFieldsFwd.H

Chapter 6 Compressible Flow

References

[1] http://openfoamwiki.net/index.php/See_the_MRF_development

[2] CFD for Underhood Modeling
http://publications.lib.chalmers.se/records/fulltext/204821/204821.pdf

[3] http://www.sourceflux.de/blog/software-design-fvoptions/

[4] Roll Motion of a Box and Interaction with Free-Surface
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/ArashEslamdoost/RollMotionofaBoxandInteractionwithFreeSurface.pdf

[5] Martin Beaudoin, Inside the mixingPlane interface
ftp://ftp.heanet.ie/disk1/sourceforge/o/op/openfoam-extend/OpenFOAM_Workshops/OFW7_2012_Darmstadt/Workshop-Documents/CommunityDay/SIG-Turbo/Final-MartinBeaudoin-OFTurboWGInsideTheMixingPlaneInterfac (accessed 06/13/2015)

[6] Ant´onio Jo˜ao Ferreira Reis, Validation of NASA Rotor 67 with OpenFOAM's Transonic Density-Based solver
http://run.unl.pt/bitstream/10362/9930/1/Reis_2013.pdf (accessed 06/13/2015)

[7] Evaluation of an improved mixing plane interface for OpenFOAM
http://publications.lib.chalmers.se/records/fulltext/201855/local_201855.pdf

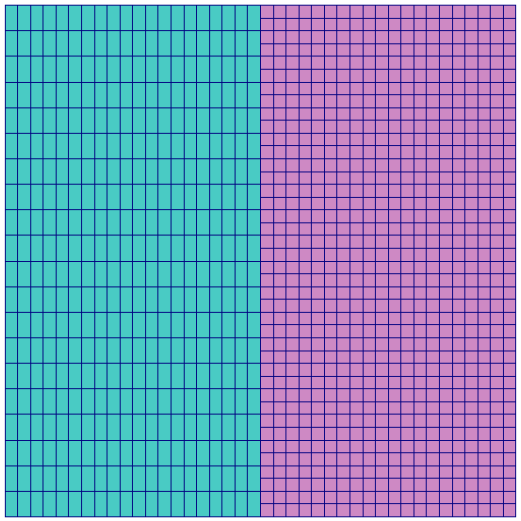
[8] Taking the mixingPlane interface for a ride
http://download2.polytechnic.edu.na/pub4/sourceforge/o/op/openfoam-extend/OpenFOAM_Workshops/OFW9_2014_Zagreb/Presentations/Hakan_Nilsson_OFW09_P_0046.pdf

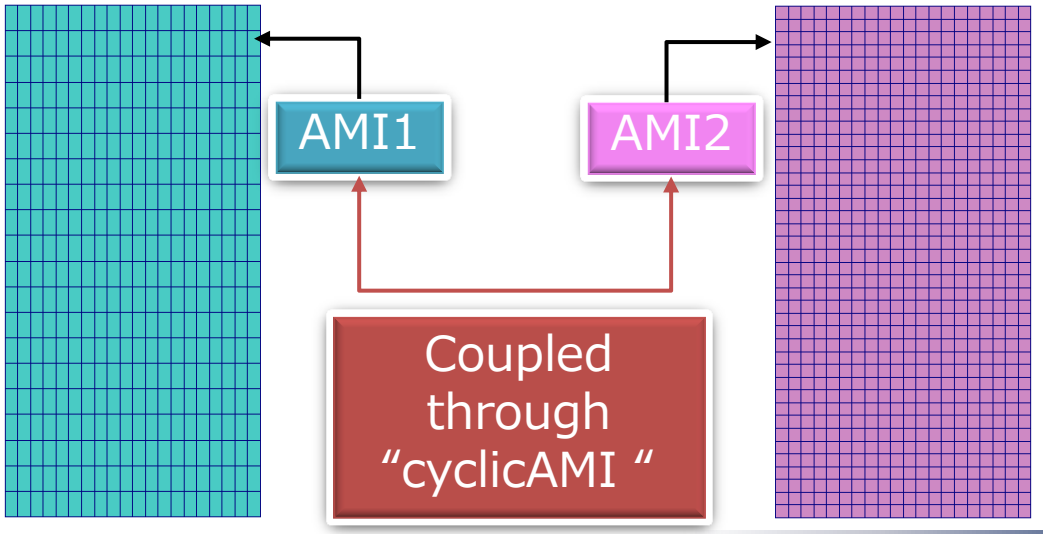
References

[9] Density Based Navier Stokes Solver for Transonic Flows
http://www.personal.psu.edu/dab143/OFW6/Presentations/oliver_borm_slides.pdf

Appendix

```
AMI1
{
    type            cyclicAMI;
    inGroups        1 (cyclicAMI);
    nFaces          20;
    startFace       2440;
    neighbourPatch  AMI2;
    transform       noOrdering;
}

AMI2
{
    type            cyclicAMI;
    inGroups        1 (cyclicAMI);
    nFaces          40;
    startFace       2460;
    neighbourPatch  AMI1;
    transform       noOrdering;
}
```





Let's check the weights!

```
AMI: Creating addressing and weights between 20 source faces and 40 target faces
AMI: Patch source sum(weights) min/max/average = 1, 1, 1
AMI: Patch target sum(weights) min/max/average = 1, 1, 1
```

sum(weights)=1 is the ideal value.
This means the patches conform perfectly.

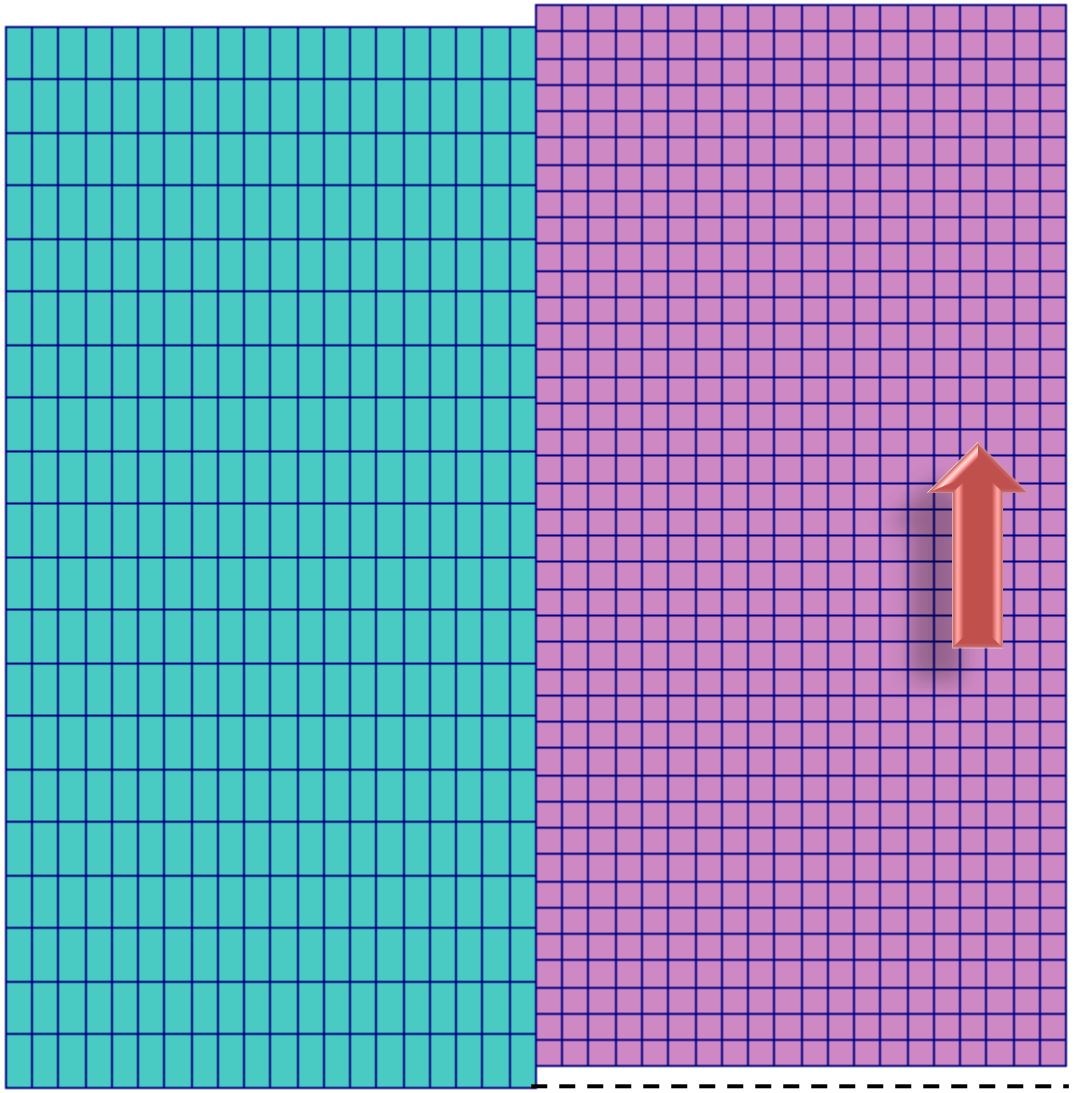
```
AMI1
{
    type            cyclicAMI;
    inGroups        1 (cyclicAMI);
    nFaces          20;
    startFace       2440;
    neighbourPatch  AMI2;
    transform       noOrdering;
}

AMI2
{
    type            cyclicAMI;
    inGroups        1 (cyclicAMI);
    nFaces          40;
    startFace       2460;
    neighbourPatch  AMI1;
    transform       noOrdering;
}
```

source patch

target patch

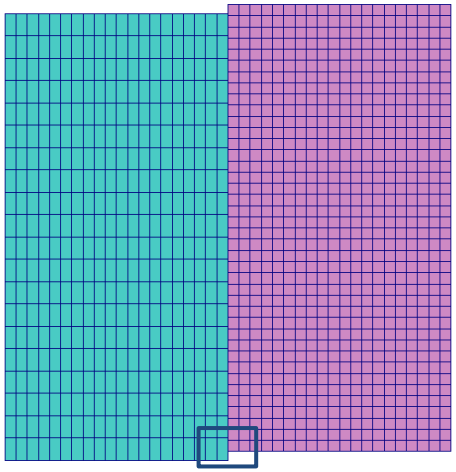
Appearing first
in "boundary" file

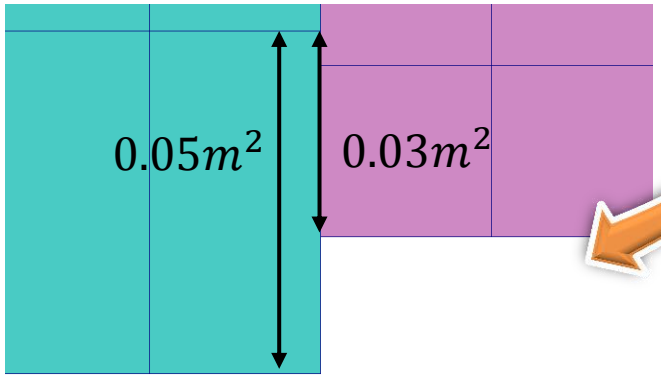
Translate the pink-colored volume upward by 0.02m



Let's recheck the weights!

AMI: Creating addressing and weights between 20 source faces and 40 target faces
AMI: Patch source sum(weights) min/max/average = 0.6, 1, 0.98
AMI: Patch target sum(weights) min/max/average = 0.2, 1, 0.98



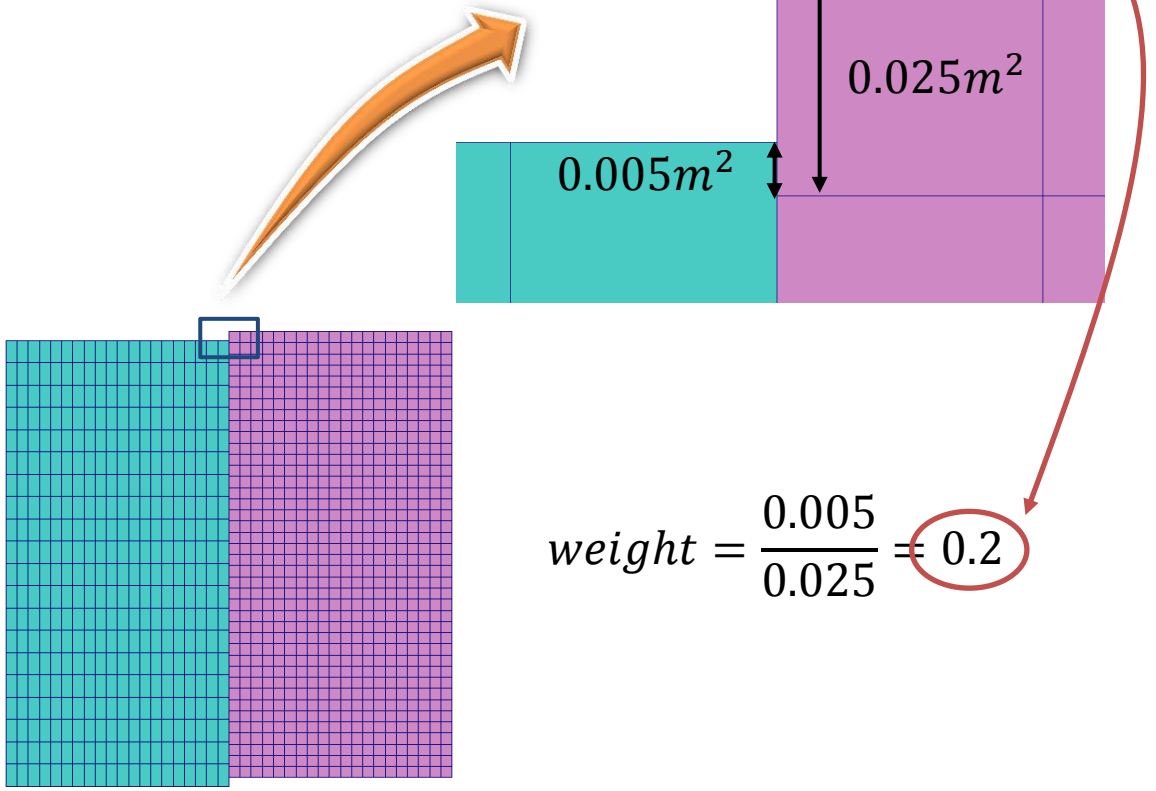


Fraction of
the intersecting areas

$$sum(weights) = \frac{0.03}{0.05} = 0.6$$

Let's recheck the weights!

AMI: Creating addressing and weights between 20 source faces and 40 target faces
AMI: Patch source sum(weights) min/max/average = 0.6, 1, 0.98
AMI: Patch target sum(weights) min/max/average = 0.2, 1, 0.98



Release note Version2.3.0

OpenFOAM® v2.3.0: Arbitrary Mesh Interface

17th February 2014

Non-Conforming AMI Patches

The arbitrary mesh interface (AMI) was introduced in OpenFOAM v2.1.0 to enable simulation across disconnected, adjacent, mesh domains. It is particularly useful for rotating geometries. Those cases require separate meshes for rotating and/or static regions of geometry, which are coupled at patch boundaries through the `cyclicAMI` boundary condition.

In previous versions of OpenFOAM, there was a requirement that the patches conform fairly closely to one another. For complex geometries, this places demands on the meshing process, especially around features, *e.g.* cylindrical end-caps. In the AMI procedure, each face accepts contributions from partially overlapping faces from the neighbour patch, with the *weights* defining the contribution as a fraction of the intersecting areas. For each face, the sum of the weights (contributions) should equal 1. Where the patch geometries are not well matched, conservation errors are introduced and the sum of weights deviates from 1. Since errors are localised to particular faces, which are often few in number, this deviation should not cause the method to fail.

Therefore, in version 2.3.0 a zero-gradient condition can be applied to patch faces whose weights are below a user-specified threshold. This is controlled by the optional `lowWeightCorrection` keyword in the mesh *boundary* file.

```
AMIPatch
{
    type            cyclicAMI;
    ...
    lowWeightCorrection 0.2;
}
```

User specified threshold for sum(weights)

Thank You!

Kindly let me know
if you have trouble downloading this slide.