A PROJECT REPORT

ON

CFD-DEM Coupling Simulations of particles using OpenFOAM & LIGGGHTS

BY

K.BADARI VISHAL 2018A4PS0807G

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

ACADEMIC YEAR 2020-21

A PROJECT REPORT

ON

CFD-DEM Coupling Simulations of particles using OpenFOAM & LIGGGHTS

BY

K.BADARI VISHAL 2018A4PS0807G B.E. (Hons.) MECHANICAL

Prepared in fulfillment of the
Design Oriented Project (DOP)
Course No. ME F376

Under

Dr. Pritanshu Ranjan
Assistant Professor
Department of Mechanical Engineering
BITS Pilani KK Birla Goa Campus

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
ACADEMIC YEAR 2020-21

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Title of the Project: Design Oriented Project on CFD-DEM coupling Simulations of particles using OpenFOAM & LIGGGHTS.

ID No.	NAME	Discipline
2018A4PS0807G	K.BADARI VISHAL	B.E. (Hons) Mechanical

Name of the Mentor: Dr. PRITANSHU RANJAN, Assistant Professor, Dept. of Mechanical Engineering, BITS Pilani K.K.Birla Goa Campus

Keywords: Discrete Element Method (DEM), CFD-DEM coupling, Partially Averaged Navier-Stokes (PANS) model, PANS modelling, PANS turbulence Model, Reynolds Averaged Simulations (RAS), Large Eddy Simulations (LES), PitzDaily, OpenFOAM, LIGGGHTS.

Abstract: This report focuses on the study & simulations performed over CFD-DEM coupling simulations of a Spouted Bed Dryer using CFD-DEM solvers such as OpenFOAM & LIGGGHTS. The current study also discusses the implementation of Partially Averaged Navier-Stokes (PANS) turbulence models using OpenFOAM-5.x followed by their respective comparisons with Reynolds Averaged Simulations RAS) & Large Eddy Simulations (LES) models on built-in “PitzDaily Case” in OpenFOAM-5.x.

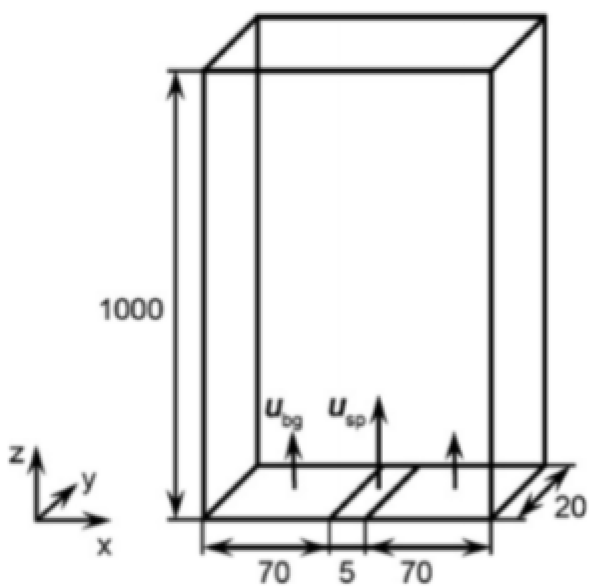
TABLE OF CONTENTS

1. CFD-DEM simulations using LIGGGHTS & OpenFOAM

1.1 Material Properties used

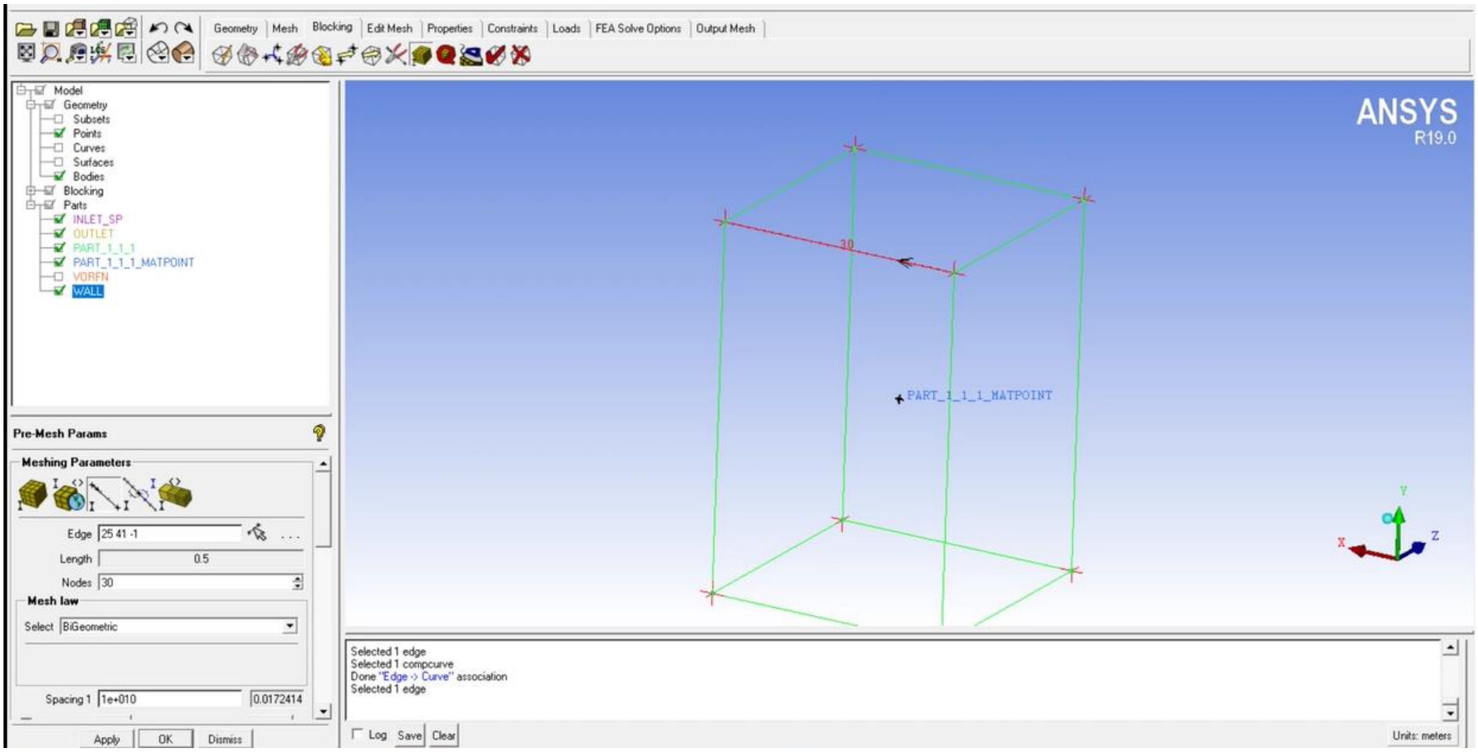
Young's Modulus	1E7 N/m^2
Gas Density	1.205 kg/m^3
Particle Density	2505 kg/m^3
Poisson's Ratio	0.45
Coefficient of Restitution	0.1
Coefficient of Friction	0.3
CFD Time Step	5e-4
DEM Time Step	1e-5

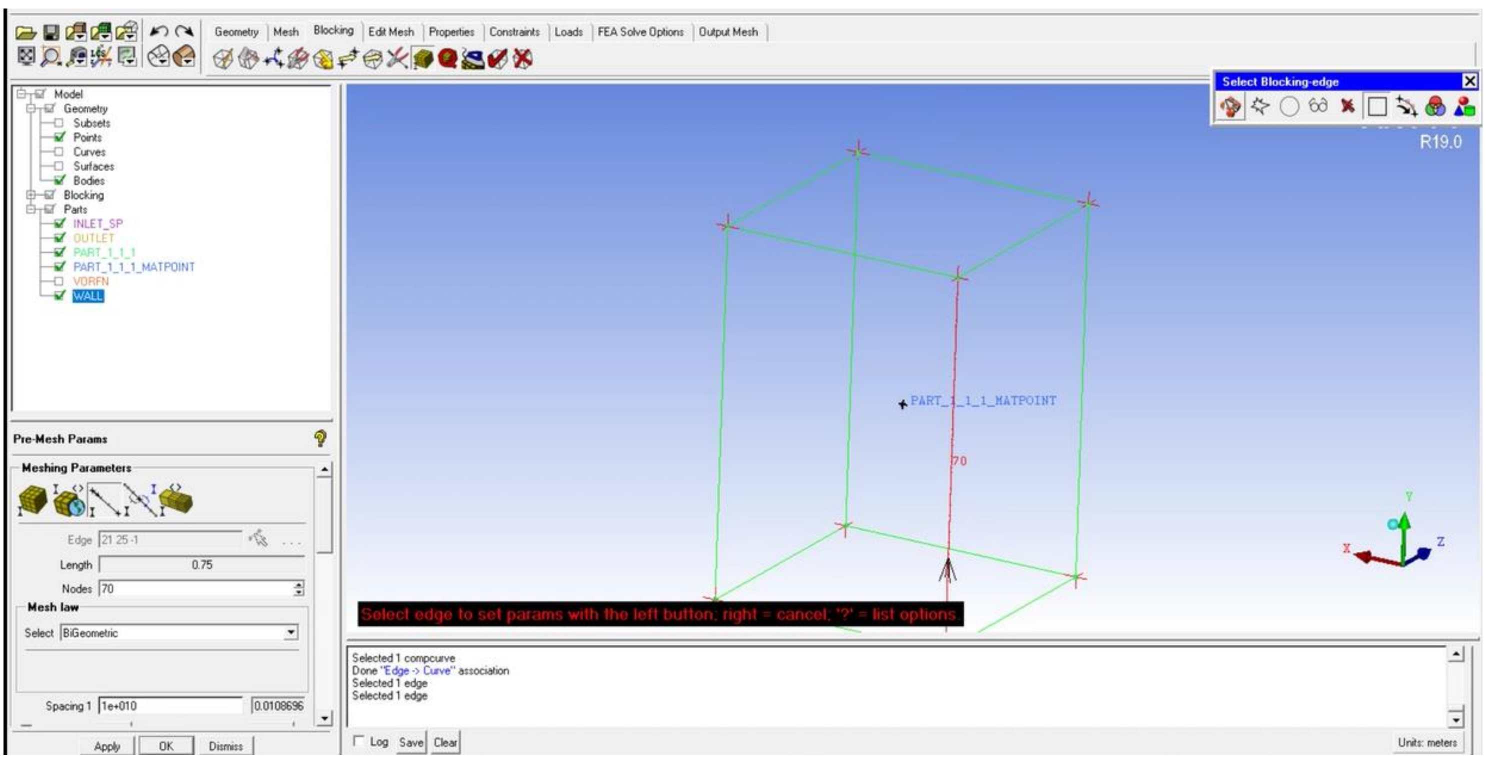
1.2 Geometry Details



1.3 Meshing Details

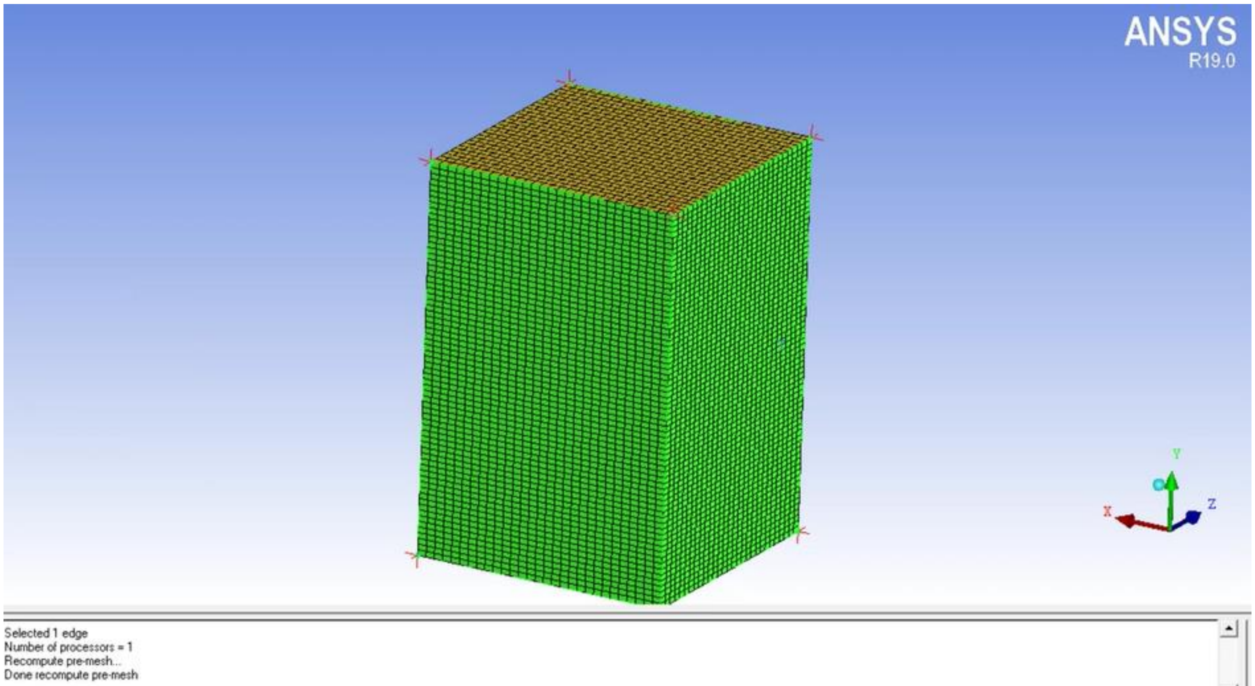
1.3.1 Node Selection:



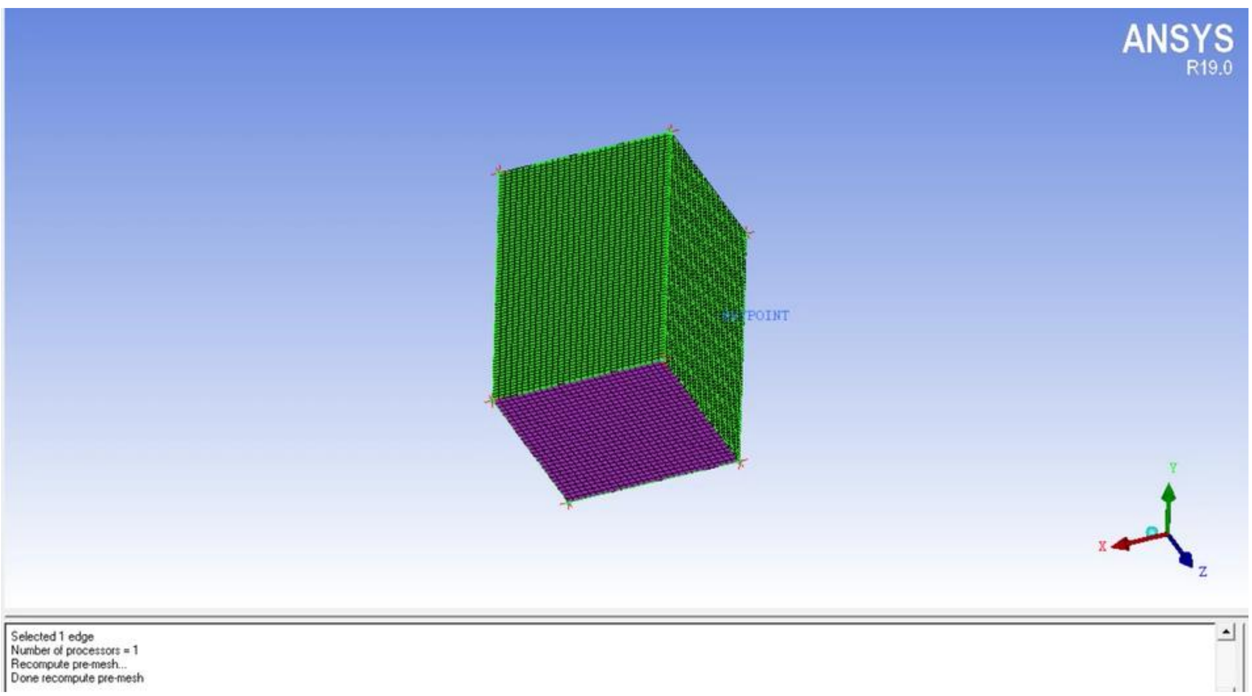


1.3.2 Mesh:

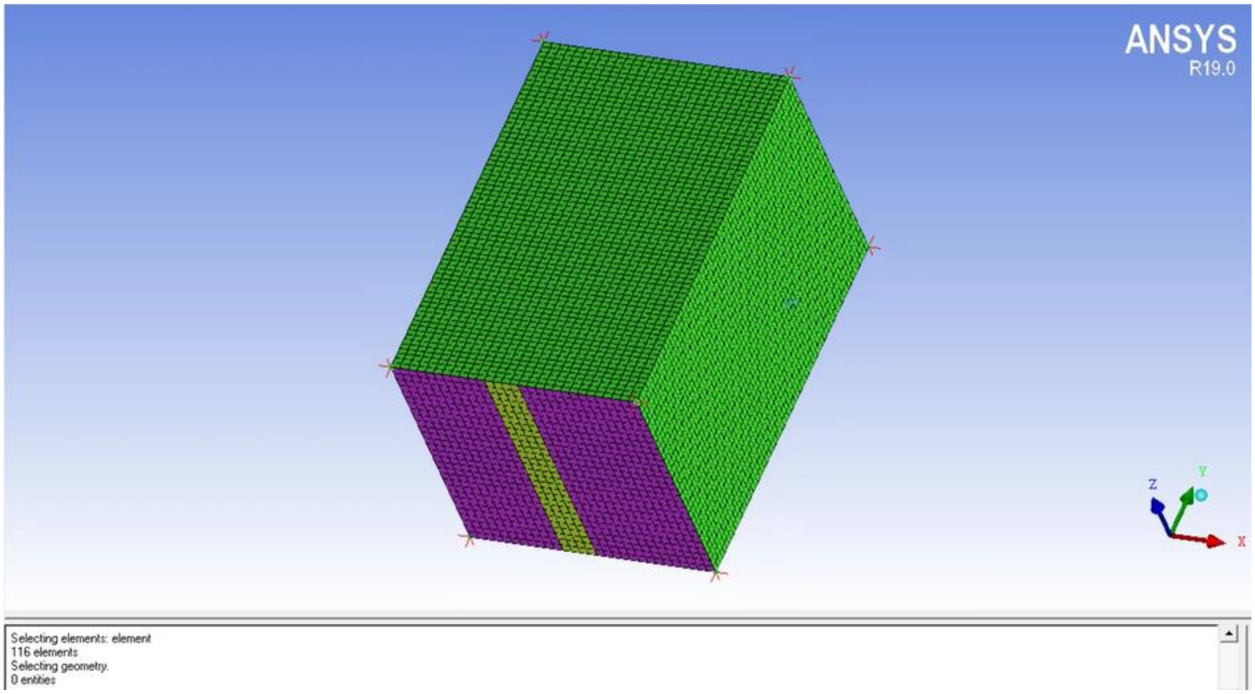
Outlet & Walls



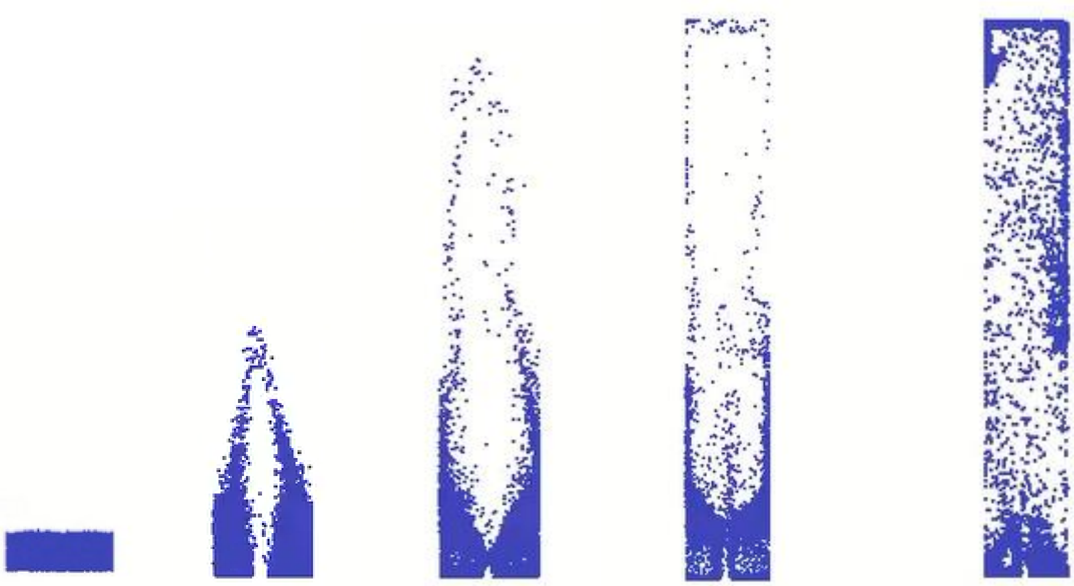
Inlet



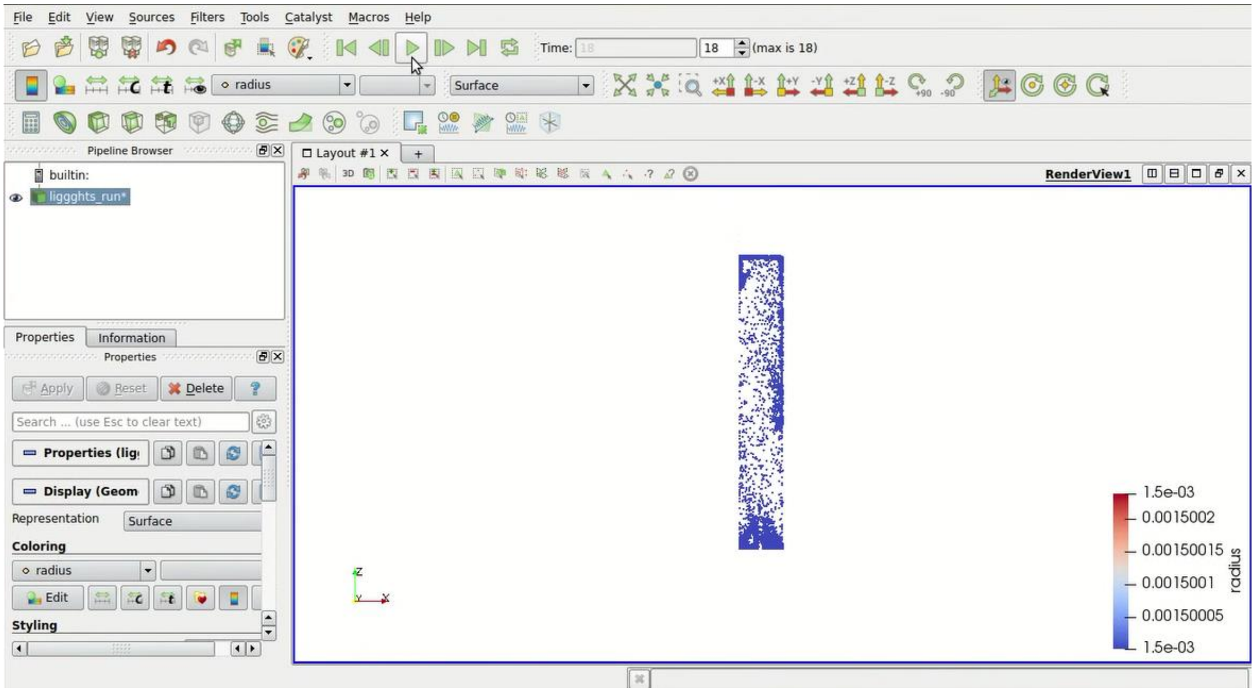
Spout Inlet & Inlet Background Fluidization Air



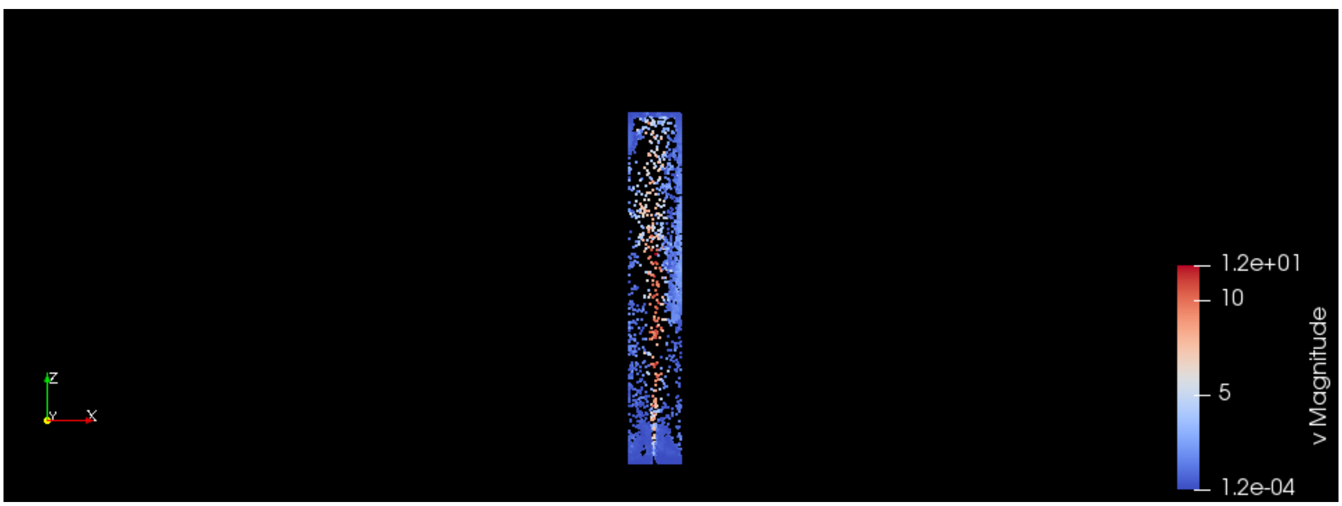
1.4 Simulation



Radius



Velocity



2. OpenFOAM-5.x PANS Model Implementation

2.1 kEpsilonPANS.C

The additional code implemented in the kEpsilon model under kEpsilon.C file in order to convert it PANS model is as shown below:

```
fEpsilon_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "fEpsilon",
        this->coeffDict_,
        1.0
    )
),

uLim_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "fKupperLimit",
        this->coeffDict_,
        1.0
    )
),

loLim_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "fKlowerLimit",
        this->coeffDict_,
        0.1
    )
),

fK_
(
    IOobject
    (
        IOobject::groupName("fK", U.group()),
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_,
    dimensionedScalar("zero",loLim_)
),

C2U
```

```
    (
        IOobject
        (
            "C2U",
            this->runTime_.timeName(),
            this->mesh_
        ),
        C1_ + (fK_/fEpsilon_)*(C2_ - C1_)
    ),

    delta_
    (
        LESdelta::New
        (
            IOobject::groupName("delta", U.group()),
            *this,
            this->coeffDict_
        )
    ),
    kU_
    (
        IOobject
        (
            IOobject::groupName("kU", U.group()),
            this->runTime_.timeName(),
            this->mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        k_*fK_,
        k_.boundaryField().types()
    ),
    epsilonU_
    (
        IOobject
        (
            IOobject::groupName("epsilonU", U.group()),
            this->runTime_.timeName(),
            this->mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        epsilon_*fEpsilon_,
        epsilon_.boundaryField().types()
    )
{
    bound(k_, this->kMin_);
    bound(epsilon_, this->epsilonMin_);
    bound(kU_, min(fK_)*this->kMin_);
    bound(epsilonU_, fEpsilon_*this->epsilonMin_);


    if (type == typeName)
    {
        this->printCoeffs(type);
        correctNut();
    }
```

```
}
```

Now we need to change the solved equations. We want to solve for unresolved quantities.

```
// Update epsilon and G at the wall
epsilonU_.boundaryFieldRef().updateCoeffs();

// Unresolved Dissipation equation
tmp<fvScalarMatrix> epsUEqn
(
    fvm::ddt(alpha, rho, epsilonU_)
  + fvm::div(alphaRhoPhi, epsilonU_)
  - fvm::laplacian(alpha*rho*DepsilonUEff(), epsilonU_)
 ==
    C1_*alpha()*rho()*G*epsilonU_()/kU_()
  - fvm::SuSp(((2.0/3.0)*C1_ + C3_)*alpha()*rho()*divU, epsilonU_)
  - fvm::Sp(C2U*alpha()*rho()*epsilonU_()/kU_(), epsilonU_)
  + epsilonSource()
  + fvOptions(alpha, rho, epsilonU_)
);

epsUEqn.ref().relax();
fvOptions.constrain(epsUEqn.ref());
epsUEqn.ref().boundaryManipulate(epsilonU_.boundaryFieldRef());
solve(epsUEqn);
fvOptions.correct(epsilonU_);
bound(epsilonU_, fEpsilon_*this->epsilonMin_);


// Unresolved Turbulent kinetic energy equation
tmp<fvScalarMatrix> kUEqn
(
    fvm::ddt(alpha, rho, kU_)
  + fvm::div(alphaRhoPhi, kU_)
  - fvm::laplacian(alpha*rho*DkUEff(), kU_)
 ==
    alpha()*rho()*G
  - fvm::SuSp((2.0/3.0)*alpha()*rho()*divU, kU_)
  - fvm::Sp(alpha()*rho()*epsilonU_()/kU_(), kU_)
  + kSource()
  + fvOptions(alpha, rho, kU_)
);

kUEqn.ref().relax();
fvOptions.constrain(kUEqn.ref());
solve(kUEqn);
fvOptions.correct(kU_);
bound(kU_, min(fK_)*this->kMin_);

// Calculation of Turbulent kinetic energy and Dissipation rate
k_ = kU_/fK_;
k_.correctBoundaryConditions();
```

```
epsilon_ = epsilonU_/fEpsilon_;
epsilon_.correctBoundaryConditions();
bound(k_, this->kMin_);
bound(epsilon_, this->epsilonMin_);

correctNut();

// Recalculate fK and C2U with new kU and epsilonU

// Calculate the turbulence integral length scale
volScalarField::Internal Lambda
(
    pow(k_,1.5)/epsilon_
);

// update fK
fK_.primitiveFieldRef() = min(max(
    sqrt(Cmu_.value())*pow(delta()/Lambda,2.0/3.0), loLim_), uLim_);

// update C2U
C2U = C1_ + (fK_/fEpsilon_)*(C2_ - C1_);
```

2.2 kEpsilonPANS.H

The additional code implemented in the kEpsilon model under kEpsilon.H file in order to convert it PANS model is as shown below:

```
// Protected data

    // Model coefficients

        dimensionedScalar Cmu_;
        dimensionedScalar C1_;
        dimensionedScalar C2_;
        dimensionedScalar C3_;
        dimensionedScalar sigmak_;
        dimensionedScalar sigmaEps_;
        dimensionedScalar fEpsilon_;
        dimensionedScalar uLim_;
        dimensionedScalar loLim_;

    // Fields

        volScalarField fK_;
        volScalarField C2U;
        //- Run-time selectable delta model
        autoPtr<Foam::LESdelta> delta_;

        volScalarField k_;
        volScalarField kU_;
```

```
            volScalarField epsilon_;
            volScalarField epsilonU_;
```

For the unresolved terms:

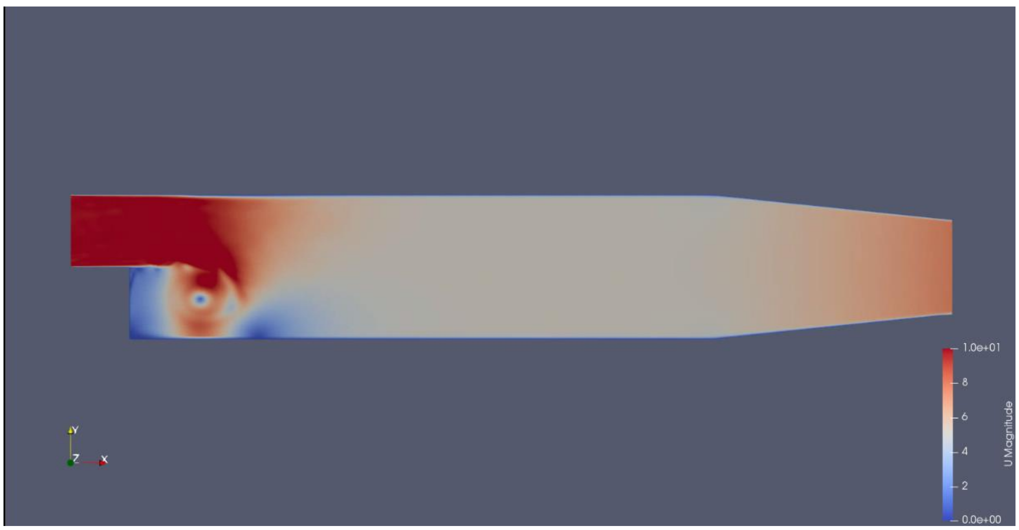
```
//- Return the unresolved turbulence kinetic energy
        virtual tmp<volScalarField> kU() const
        {
            return kU_;
        }

        //- Return the unresolved turbulence kinetic energy dissipation rate
        virtual tmp<volScalarField> epsilonU() const
        {
            return epsilonU_;
        }

        //- Access function to filter width
        inline const volScalarField& delta() const
        {
            return delta_();
        }

        //- Return the effective diffusivity for unresolved k
        tmp<volScalarField> DkUEff() const
        {
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "DkUEff",
                    (this->nut_/(fK_*fK_*sigmak_/fEpsilon_)
                     + this->nu())
                )
            );
        }

        //- Return the effective diffusivity for unresolved epsilon
        tmp<volScalarField> DepsilonUEff() const
        {
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "DepsilonUEff",
                    (this->nut_/(fK_*fK_*sigmaEps_/fEpsilon_)
                     + this->nu())
                )
            );
        }
```

3. Results

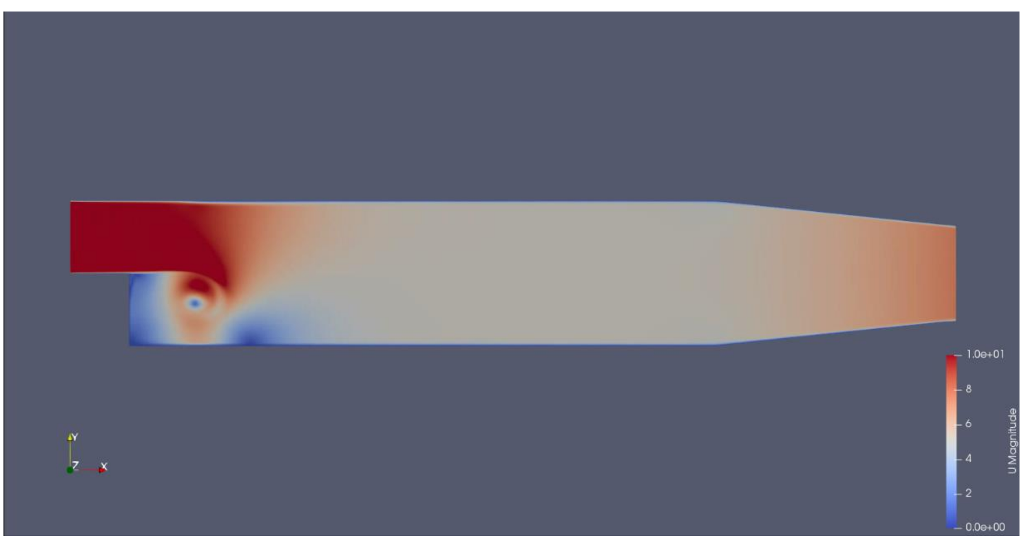
Taking the case of PitzDaily and comparing the implemented PANS model with LES simulations of PitzDaily, we arrive at the following results of the Velocity variation at various time steps:

Time step - 1

LES Model:

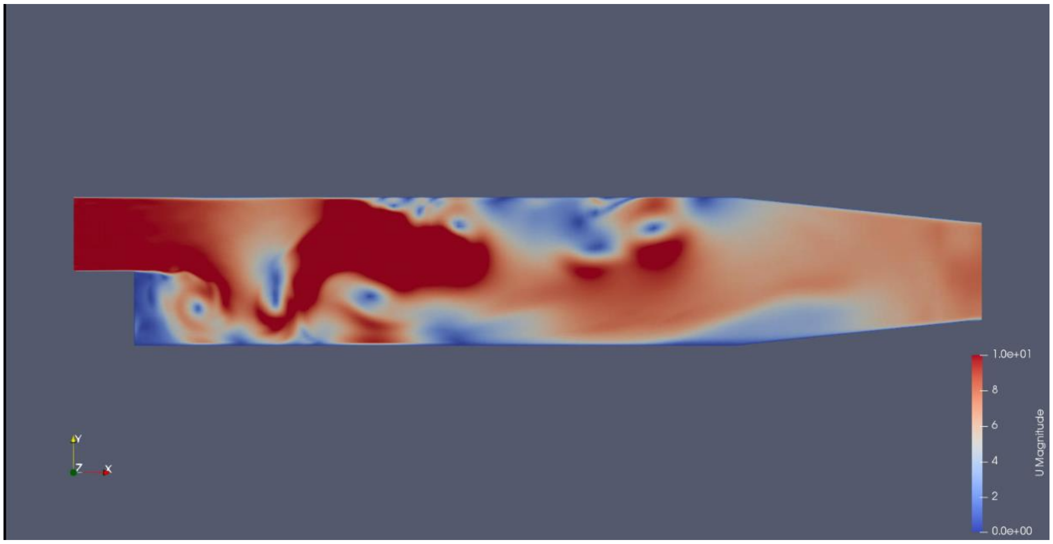


PANS Model:

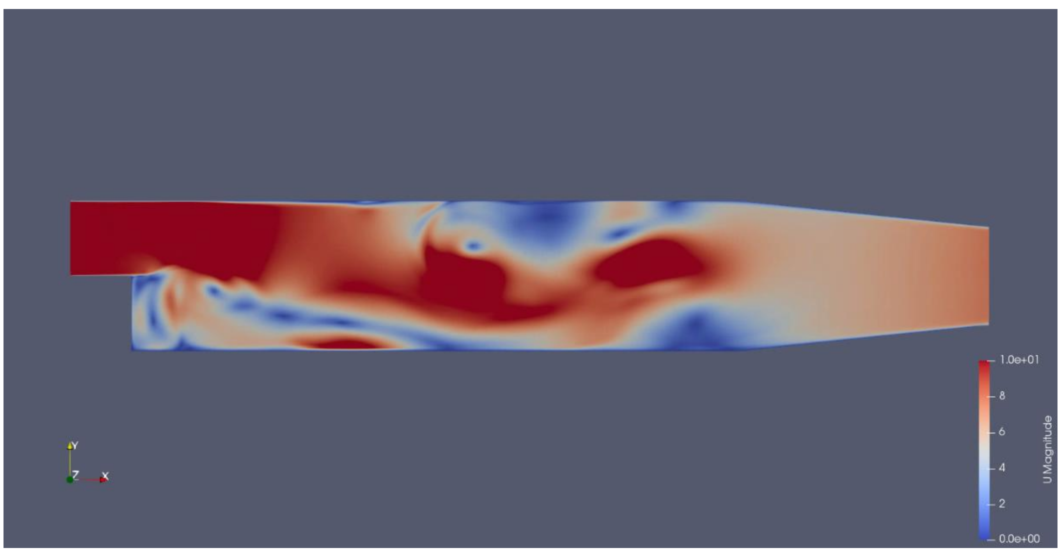


<u>**Time step – 2**</u>

LES Model:

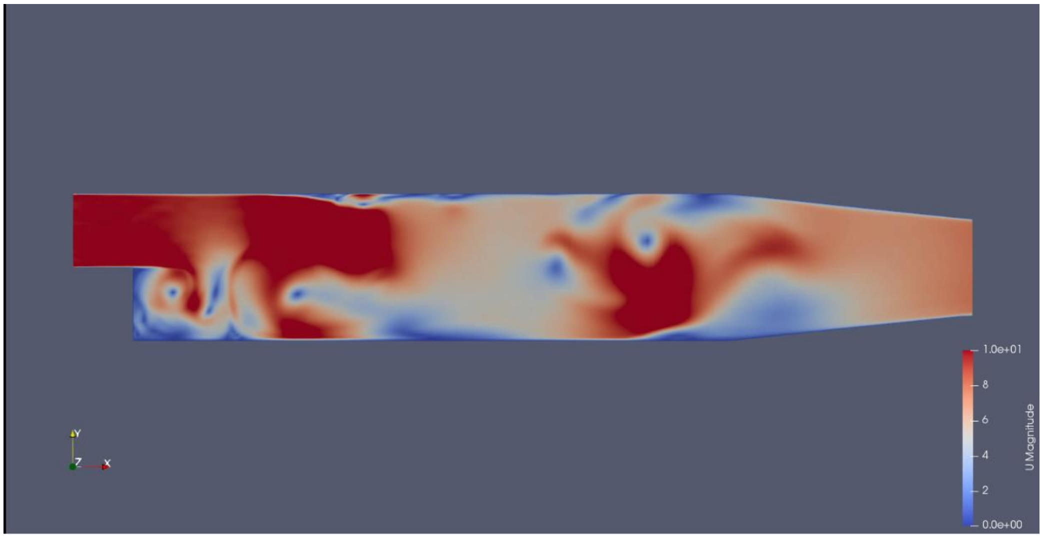


PANS Model:

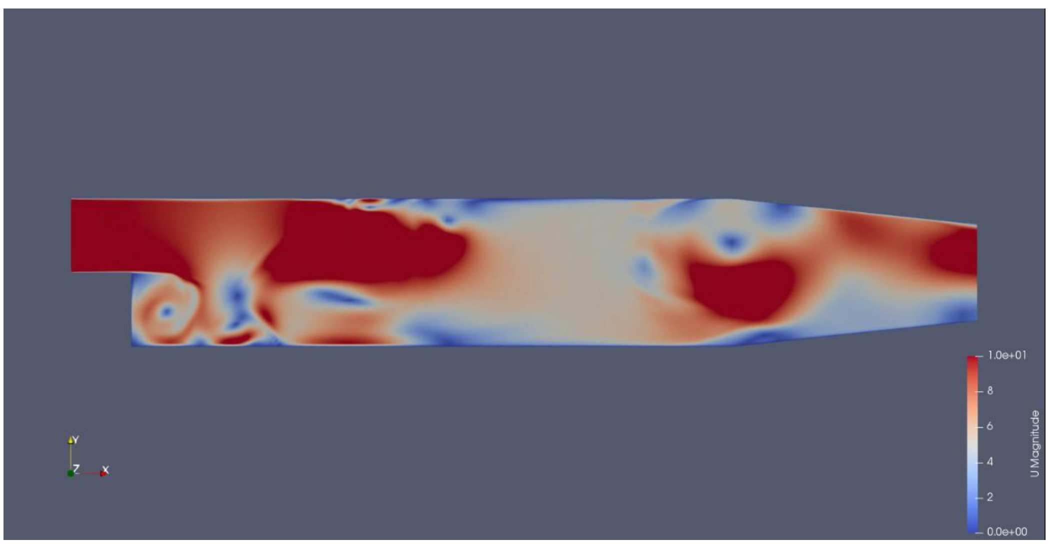


Time step – 3

LES Model:



PANS Model:



4. References

1. Norouzi, H., Zarghami, R., Sotudeh-Gharebagh, R., & Mostoufi, N. “Coupled CFD-DEM modelling - Formulation, Implementation and Application to Multiphase Flows”.
2. Matuttis, HG. & Chen, J., “UNDERSTANDING THE DISCRETE ELEMENT METHOD SIMULATION OF NON-SPHERICAL PARTICLES FOR GRANULAR AND MULTI-BODY SYSTEMS”.
3. Guglielmo Minelli, “PANS turbulence model implementation - Developed for OpenFOAM-2.3.x”.
4. Qianyi Chen, Ting Xiong, Xinzhuo Zhang, & Pan Jiang, “Study of the hydraulic transport of non-spherical particles in a pipeline based on the CFD-DEM”.