2016 5th OpenFOAM Korea User's Community Conference

# 자연대류를 포함한 열전달 해석 솔버 개선

## buoyantSimpleFoam

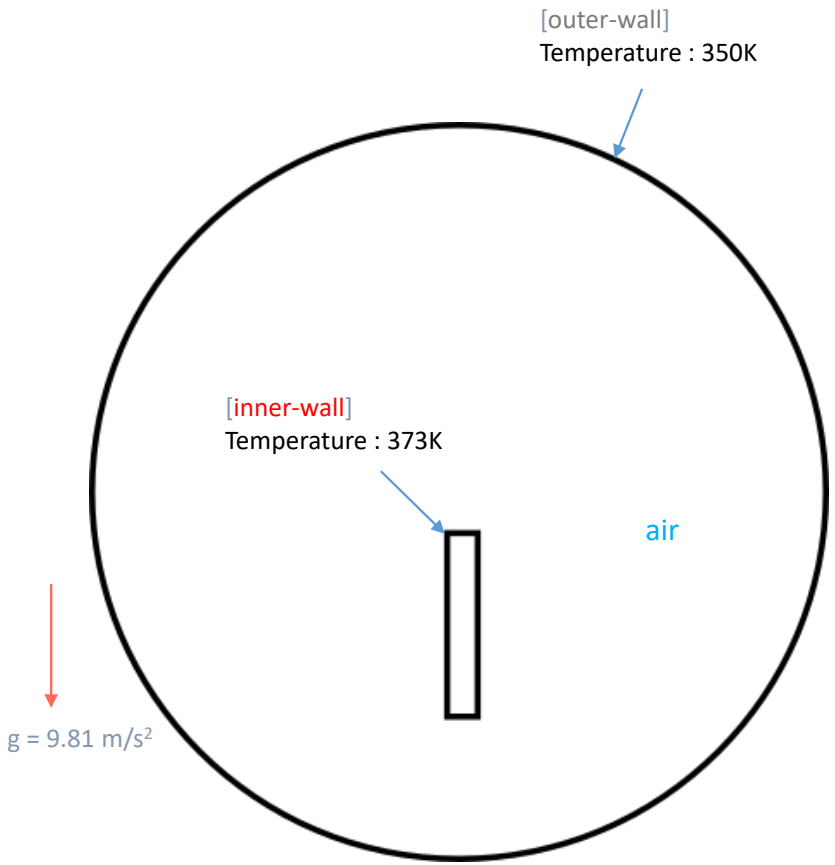
길재흥
넥스트폼 기술연구소


2016년 9월 29일

# 문제 제기

# 간단한 예제

- Natural Convection in Closed Domain

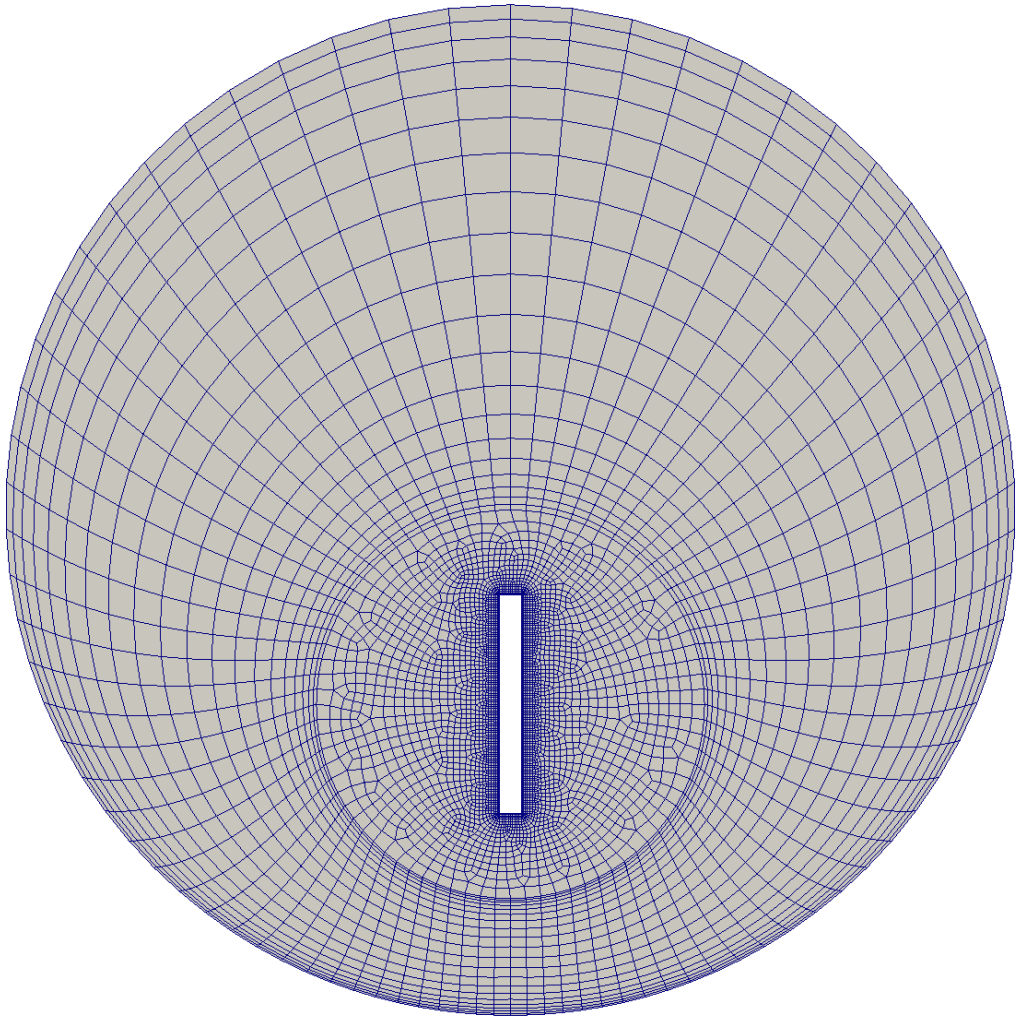



**[air properties]**

- density : perfect gas
- specific heat : constant
- viscosity : Sutherland's law
- conductivity : kinetic theory

# 간단한 예제

- Grid



# buoyantSimpleFoam in OpenFOAM-4.0(2.4)

- Pressure Boundary Conditions : $p$
  - thermophysical properties를 update하기 위한 pressure

0/p

```
.
.
.

internalField       uniform 101325;

boundaryField
{
    outer-wall
    {
        type            calculated;
        value           $internalField;
    }

    inner-wall
    {
        type            calculated;
        value           $internalField;
    }

    .
    .
    .
```

# buoyantSimpleFoam in OpenFOAM-4.0(2.4)

- Pressure Boundary Conditions : $p_{rgh} = p - \rho(\vec{g} \cdot \vec{r})$
  - 실제 계산에 사용되는 pressure(working pressure)

0/p_rgh

```
.
.
.

internalField       uniform 101325;

boundaryField
{
    outer-wall
    {
        type            fixedFluxPressure;
        value           $internalField;
    }

    inner-wall
    {
        type            fixedFluxPressure;
        value           $internalField;
    }

    .
    .
    .
```

# buoyantSimpleFoam in OpenFOAM-4.0(2.4)

- Momentum Equation

$$\nabla \cdot (\rho \vec{U} \vec{U}) = \nabla \cdot \bar{\bar{\tau}} - \nabla p - \rho \vec{g}$$

$$\boldsymbol{\nabla \cdot (\rho \vec{U} \vec{U}) = \nabla \cdot \bar{\bar{\tau}} - \nabla p_{rgh} - (\vec{g} \cdot \vec{r}) \nabla \rho}$$

$\vec{g}$ : *graivity vector*

$\bar{\bar{\tau}}$ : *shear stress tensor*

$$\bar{\bar{\tau}} = \mu_{eff} \left\{ \nabla \vec{U} + (\nabla \vec{U})^T - \frac{2}{3} (\nabla \cdot \vec{U}) \bar{\bar{I}} \right\}$$

$\mu_{eff} = \mu + \mu_t$ : *effective dynamic viscosity*

$$p_{rgh} = p - \rho (\vec{g} \cdot \vec{r})$$

# buoyantSimpleFoam in OpenFOAM-4.0(2.4)

- Pressure-Velocity Coupling
  - Use pseudo-velocity($\vec{U}_P^*$) to derive pressure equation

$$\vec{U}_P^* = \frac{H(\vec{U}^*)}{a_P} - \frac{V_P}{a_P}\{(\vec{g}\cdot\vec{r})\nabla\rho\}_P$$

$$\vec{U}_P = \vec{U}_P^* - \frac{V_P}{a_P}(\nabla p_{rgh})_P \tag{1}$$

$$\dot{m}_f = \rho_f \left\{\frac{H(\vec{U}^*)}{a}\right\}_f \cdot \vec{S}_f \underbrace{-\rho_f \left(\frac{V}{a}\right)_f (\vec{g}\cdot\vec{r})_f \left(\frac{\partial\rho}{\partial n}\right)_f |\vec{S}_f|}_{\phi_g} - \rho_f \left(\frac{V}{a}\right)_f \left(\frac{\partial p_{rgh}}{\partial n}\right)_f |\vec{S}_f| \tag{2}$$

  - Substituting equation (2) into discretized continuity equation($\sum \dot{m}_f = 0$) yields pressure equation

# buoyantSimpleFoam in OpenFOAM-4.0(2.4)

- `fixedFluxPressure`
  - $\phi_g$에 대한 flux correction

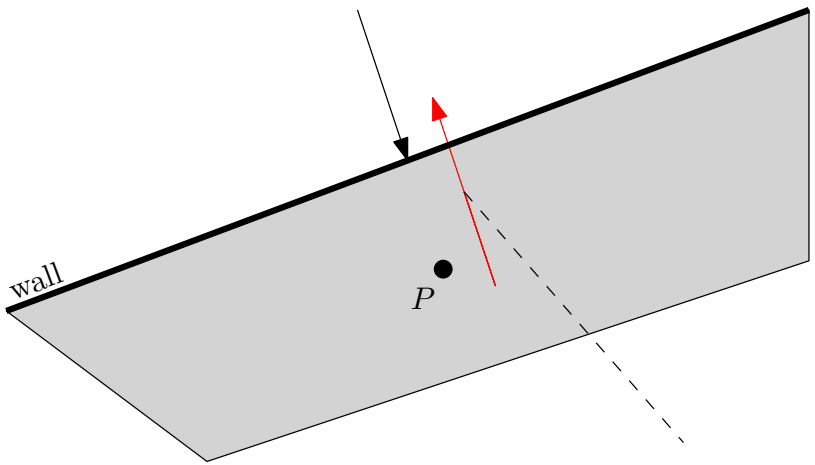
**pEqn.H**

```
    .
    .
    .
volScalarField rAU("rAU", 1.0/UEqn().A());
surfaceScalarField rhorAUf("rhorAUf", fvc::interpolate(rho*rAU));

volVectorField HbyA("HbyA", U);
HbyA = rAU*UEqn().H();
UEqn.clear();

surfaceScalarField phig(-rhorAUf*ghf*fvc::snGrad(rho)*mesh.magSf());

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

fvOptions.makeRelative(fvc::interpolate(rho), phiHbyA);

bool closedVolume = adjustPhi(phiHbyA, U, p_rgh);

phiHbyA += phig;

    .
    .
    .
```

# buoyantSimpleFoam in OpenFOAM-4.0(2.4)

- `fixedFluxPressure`
  - pressure laplacian 이산화과정에서 surface normal gradient가 $\phi_g$를 상쇄하도록

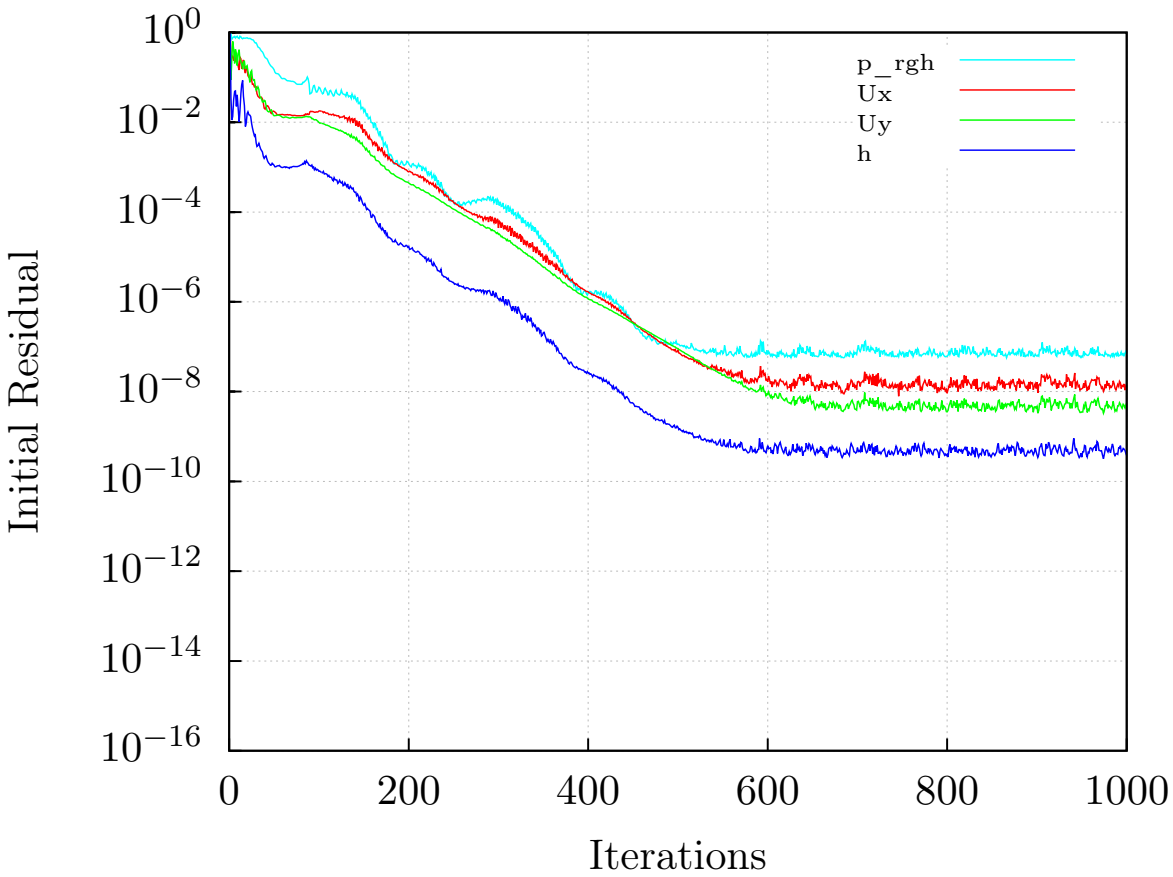
$$\phi_g = -\rho_f \left(\frac{V}{a}\right)_f (\vec{g}\cdot\vec{r})_f \left(\frac{\partial \rho}{\partial n}\right)_f |\vec{S}_f|$$




$$\text{surface normal pressure gradient} : -\frac{\partial (p_{rgh})}{\partial n} = (\vec{g}\cdot\vec{r})_f \left(\frac{\partial \rho}{\partial n}\right)_f$$

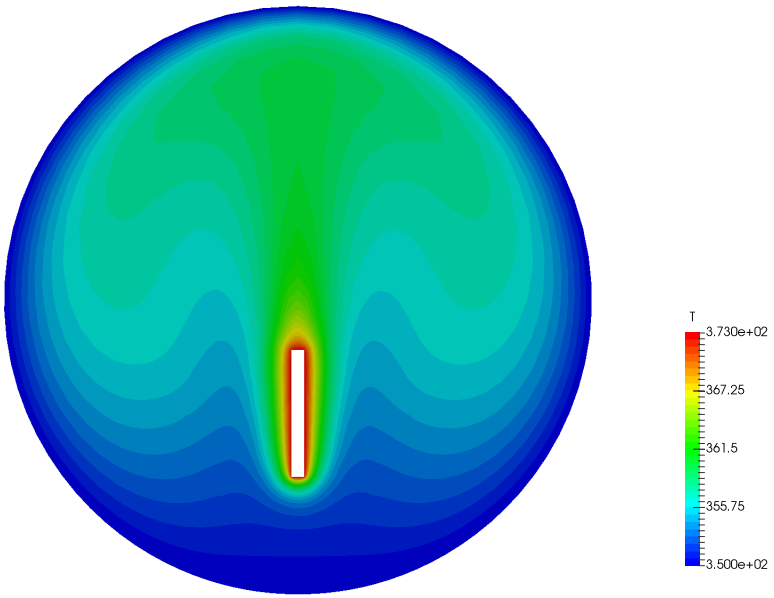
# 해석 결과

- 수렴성

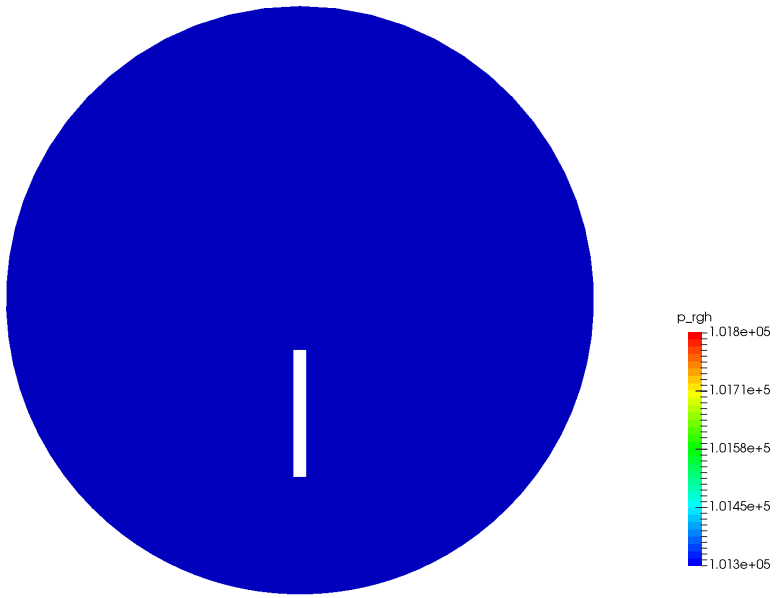

Initial Residual
Iterations
p_rgh
Ux
Uy
h
$10^{0}$
$10^{-2}$
$10^{-4}$
$10^{-6}$
$10^{-8}$
$10^{-10}$
$10^{-12}$
$10^{-14}$
$10^{-16}$
0
200
400
600
800
1000

# 해석 결과

- 온도 및 압력



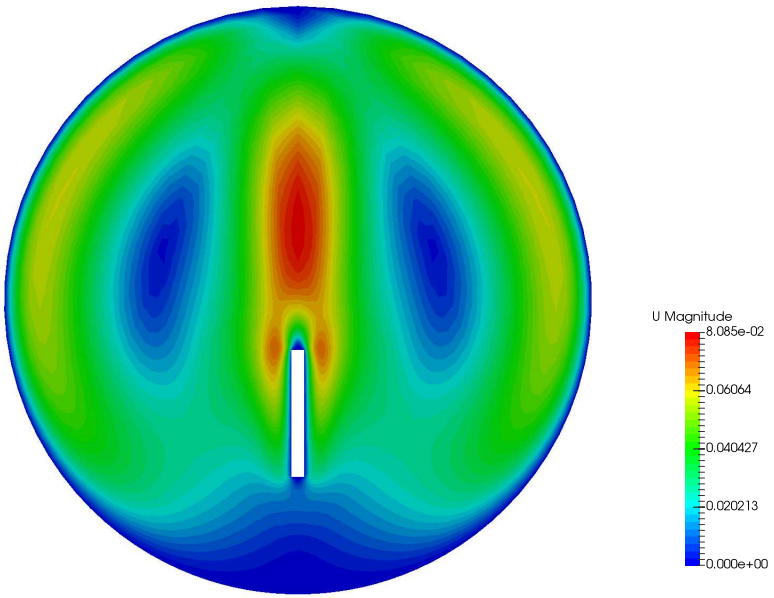

(a) 온도 분포



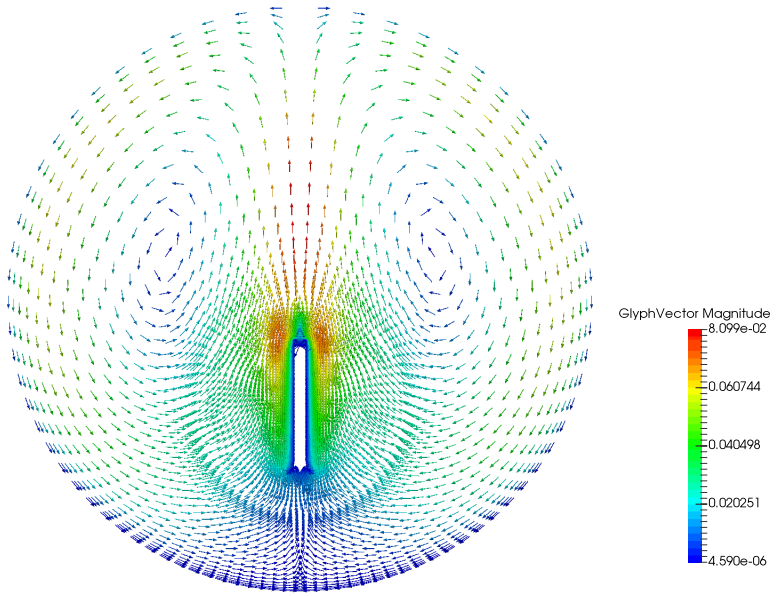

(b) 압력 분포(p_rgh)

# 해석 결과

- 속도



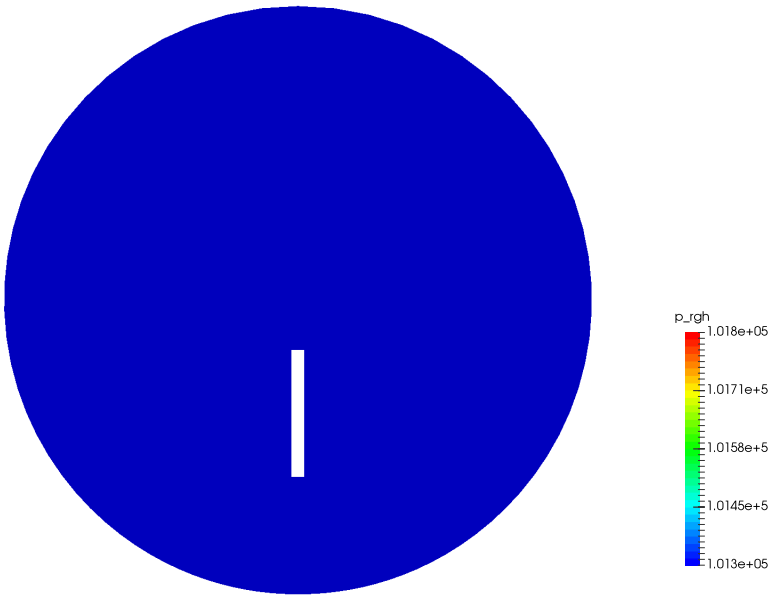

(a) 속도 분포



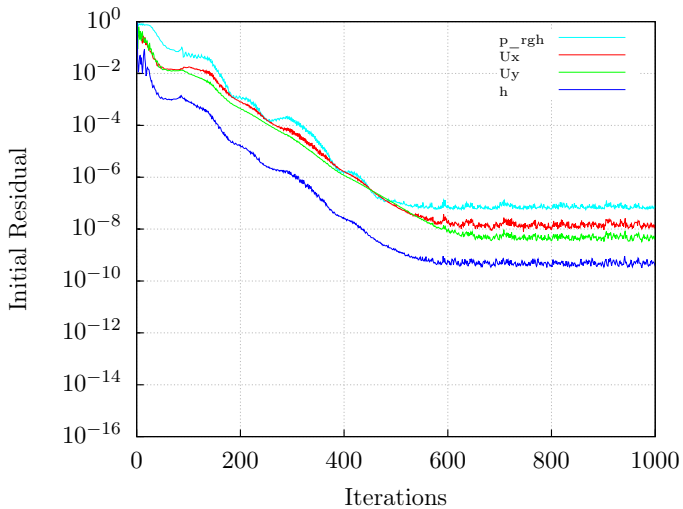

(b) 속도 벡터

# 문제점

- 압력 분포를 확인할 수 없음
- 수렴성(?)




(a) 압력 분포




(b) 수렴성

# Operating Pressure

# Operating Pressure

- 압력변화가 매우 작은 유동
  - $\Delta p$ 계산시 round-off error 누적
  - 일정 수준의 압력(operating pressure)에 대한 상대압력을 이용하면 round-off error를 줄일수 있다.

$$p_{rgh} = p - \rho(\vec{g} \cdot \vec{r}) - p^{op}$$

**pEqn.H**

```
.
.
.
            // Correct the momentum source with the pressure gradient flux
            // calculated from the relaxed pressure
            U = HbyA + rAU*fvc::reconstruct((phig - p_rghEqn.flux())/rhorAUf);
            U.correctBoundaryConditions();
            fvOptions.correct(U);
        }
    }

    #include "continuityErrs.H"

    p = p_rgh + rho*gh + Op;

.
.
.
```

# Operating Pressure

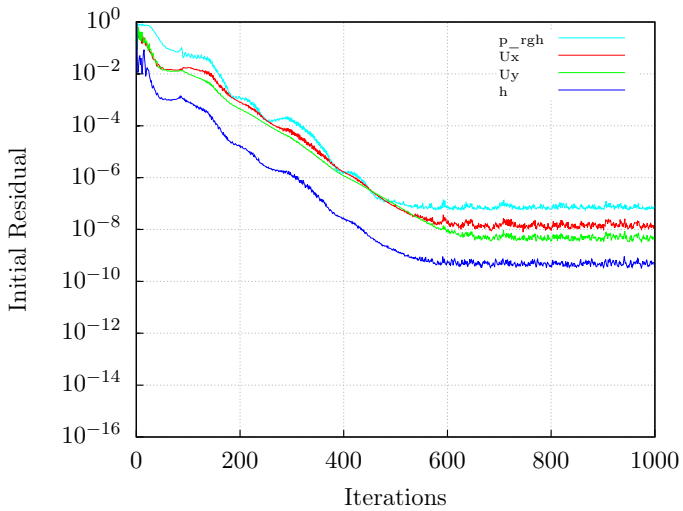
- Specifying Operating Pressure

**createFields.H**

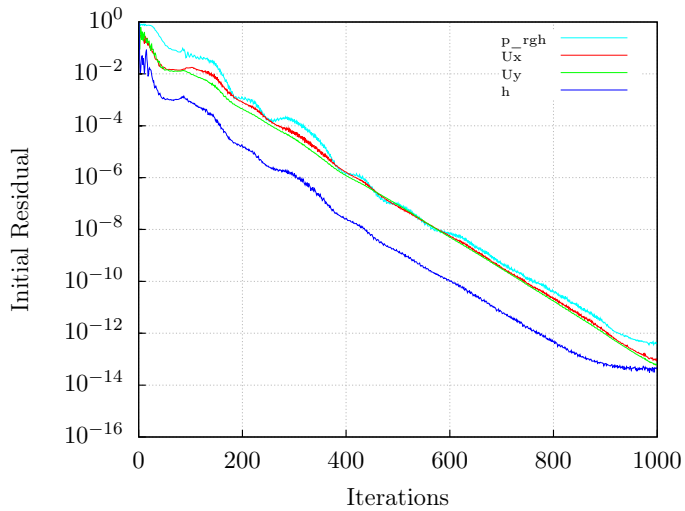
```
.
.
.
IOdictionary operatingConditions
(
    IOobject
    (
        "operatingConditions",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);

dimensionedScalar Op(operatingConditions.lookup("Op"));

.
.
.
```

# Operating Pressure

- 수렴성



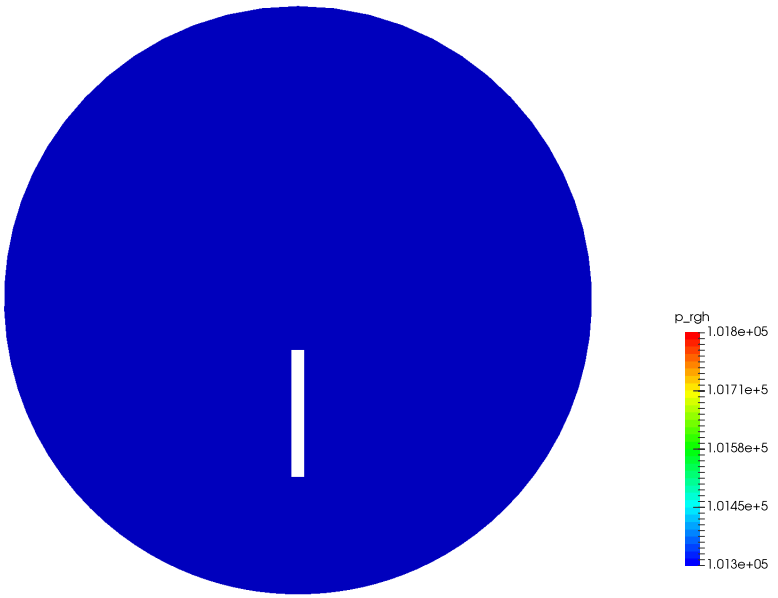

(a) $p^{op} = 0$



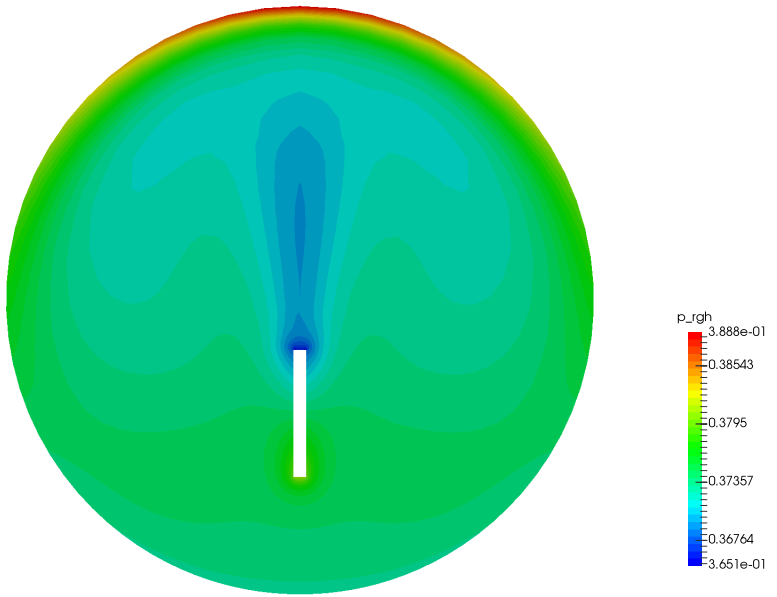

(b) $p^{op} = 101325$

# Operating Pressure

- 압력($p\_rgh$)




(a) $p^{op} = 0$




(b) $p^{op} = 101325$

# Operating Density

# Re-define Working Pressure($p_{rgh}$)

- OpenFOAM

$$p_{rgh} = p - \rho(\vec{g} \cdot \vec{r})$$

- Commercial Codes

$$p_{rgh} = p - \rho_0(\vec{g} \cdot \vec{r})$$

$\rho_0$ : average density or specified as constant

# Re-define Working Pressure($p_{rgh}$)

- Resulting Momentum Equation
  - OpenFOAM

$$\nabla \cdot (\rho \vec{U} \vec{U}) = \nabla \cdot \bar{\bar{\tau}} - \nabla p_{rgh} - (\vec{g} \cdot \vec{r}) \nabla \rho$$

  - Commercial Codes

$$\nabla \cdot (\rho \vec{U} \vec{U}) = \nabla \cdot \bar{\bar{\tau}} - \nabla p_{rgh} + (\rho - \rho_0) \vec{g}$$

# Implementation of New Momentum Equation

- Old

**UEqn.H**

```
// Solve the Momentum equation

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

UEqn().relax();

fvOptions.constrain(UEqn());

if (simple.momentumPredictor())
{
    solve
    (
        UEqn()
     ==
        fvc::reconstruct
        (
            (
              - ghf*fvc::snGrad(rho)
              - fvc::snGrad(p_rgh)
            )*mesh.magSf()
        )
    );

    fvOptions.correct(U);
}
```

- New

**UEqn.H**

```
// Solve the Momentum equation

tmp<fvVectorMatrix> UEqn
(
    fvm::div(phi, U)
  + turbulence->divDevRhoReff(U)
 ==
    (rho - rho0)*g
  + fvOptions(rho, U)
);

UEqn().relax();

fvOptions.constrain(UEqn());

if (simple.momentumPredictor())
{
    solve(UEqn() == -fvc::grad(p_rgh));

    fvOptions.correct(U);
}
```

# Implementation of New Momentum Equation

- Modify $\phi_g$
  - $\phi_g$ corresponding to body force $-(\vec{g}\cdot\vec{r})\nabla\rho$

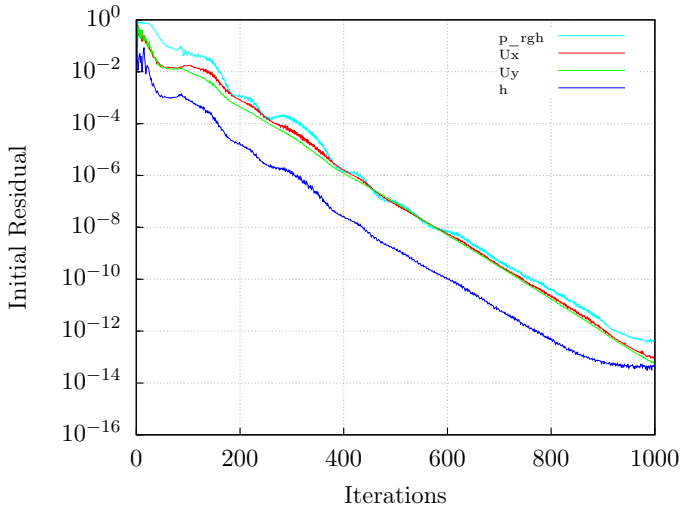
$$\phi_g = -\rho_f \left(\frac{V}{a}\right)_f (\vec{g}\cdot\vec{r})_f \left(\frac{\partial\rho}{\partial n}\right)_f |\vec{S}_f|$$

  - $\phi_g$ corresponding to body force $(\rho-\rho_0)\vec{g}$

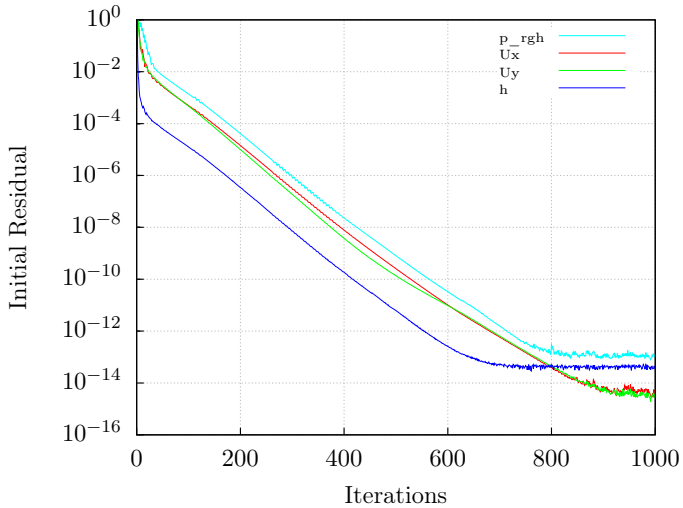
$$\phi_g = \rho_f \left(\frac{V}{a}\right)_f (\rho-\rho_0)_f (\vec{g}\cdot\vec{S}_f)$$

# Results

- Residual
  - $\rho_0$ : volume average



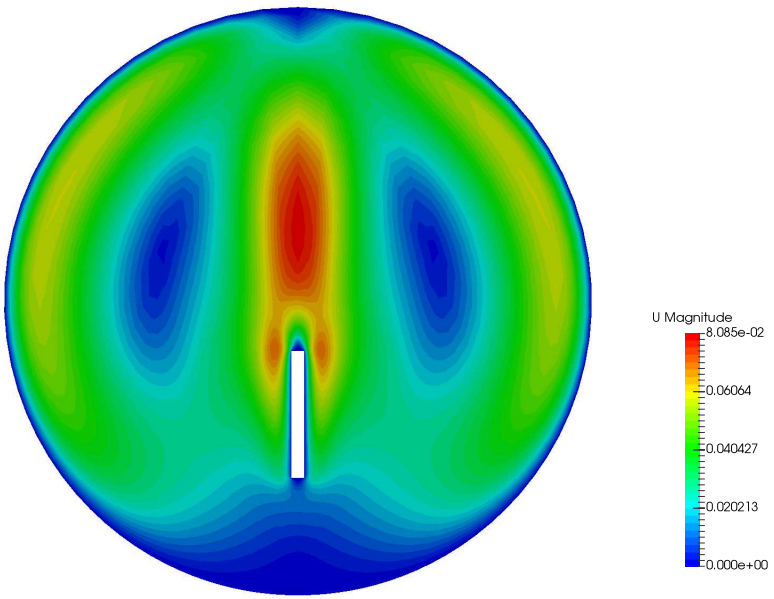

(a) $p_{rgh} = p - \rho(\vec{g} \cdot \vec{r})$



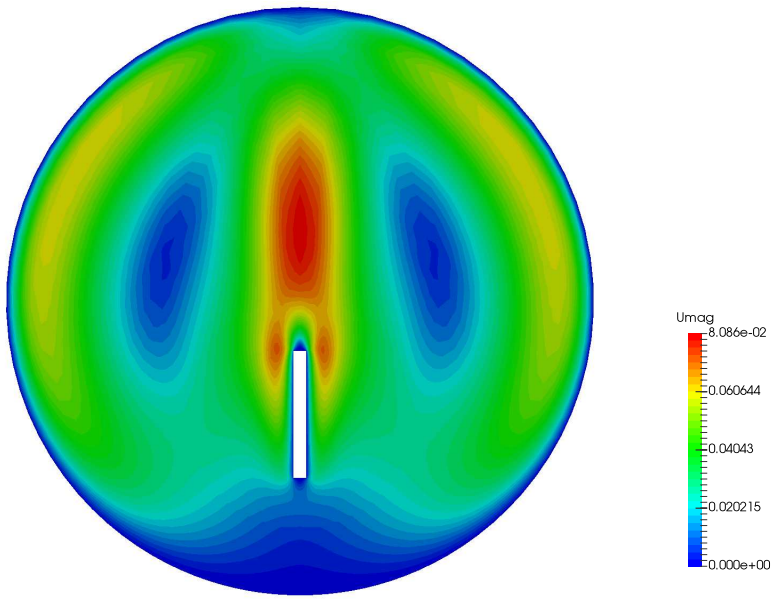

(b) $p_{rgh} = p - \rho_0(\vec{g} \cdot \vec{r})$

# Results

- Velocity Magnitude



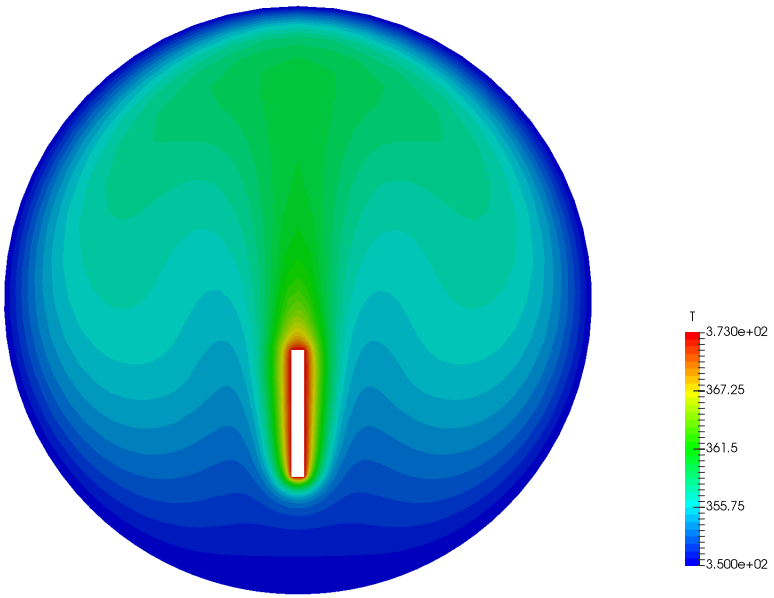

(a) $p_{rgh} = p - \rho(\vec{g} \cdot \vec{r})$



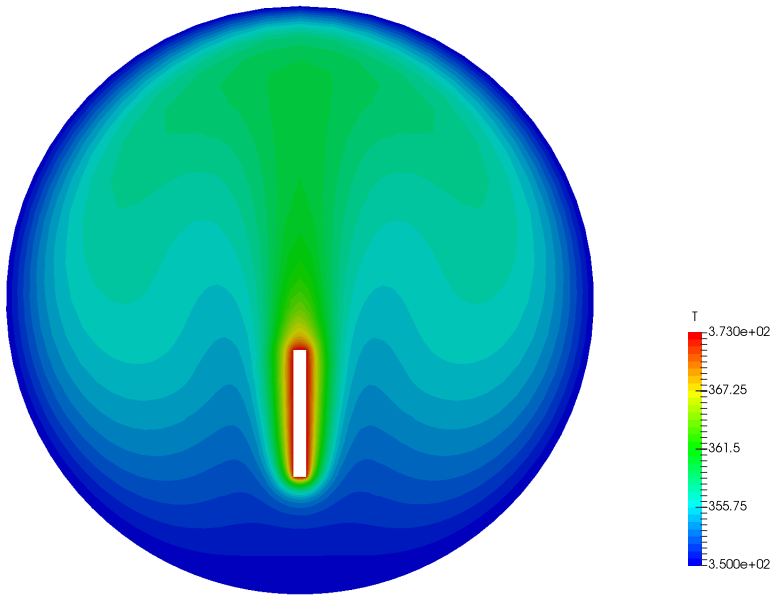

(b) $p_{rgh} = p - \rho_0(\vec{g} \cdot \vec{r})$

# Results

- Temperature



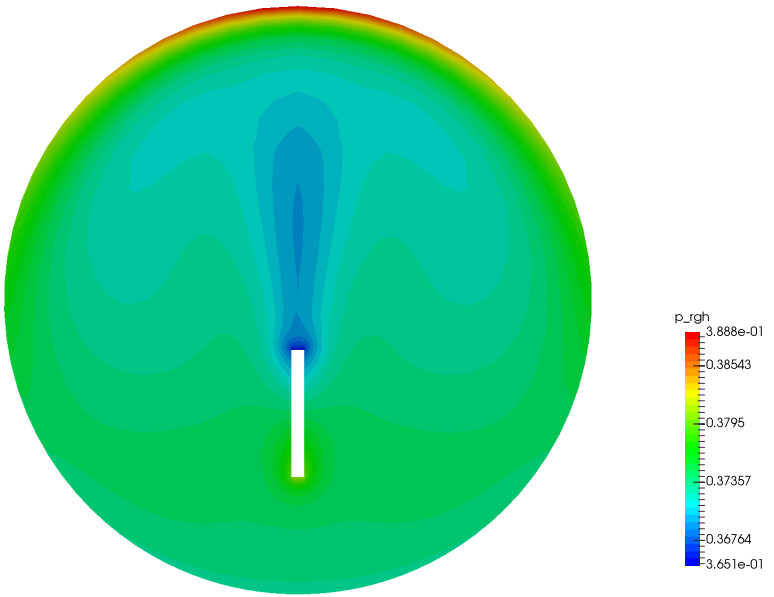

(a) $p_{rgh} = p - \rho(\vec{g} \cdot \vec{r})$



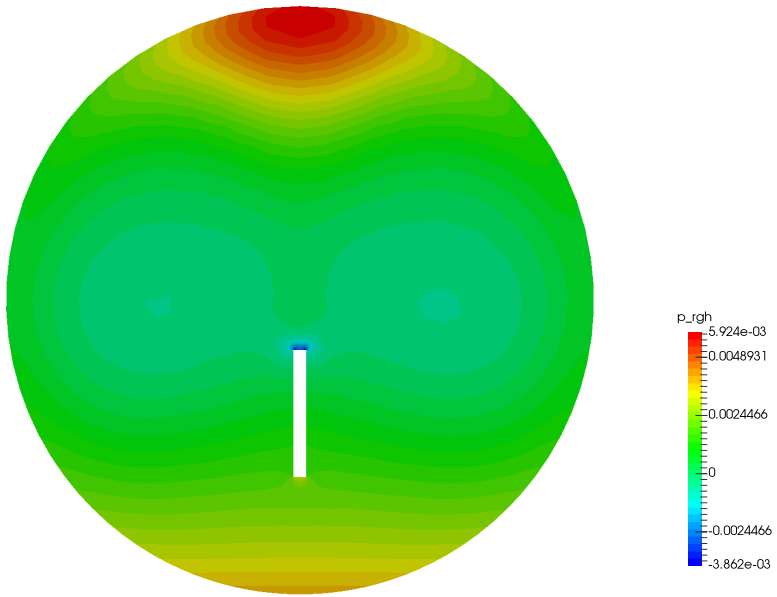

(b) $p_{rgh} = p - \rho_0(\vec{g} \cdot \vec{r})$

# Results

- Pressure($p\_rgh$)



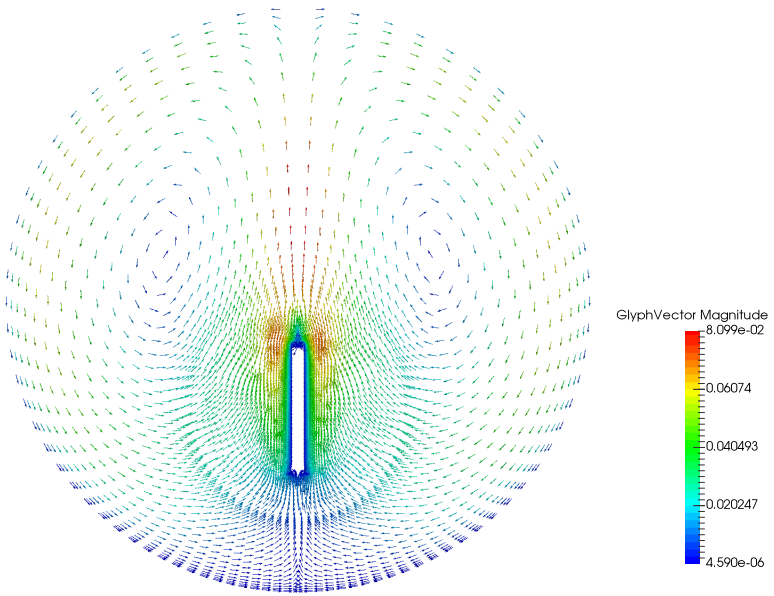

(a) $p_{rgh} = p - \rho(\vec{g} \cdot \vec{r})$



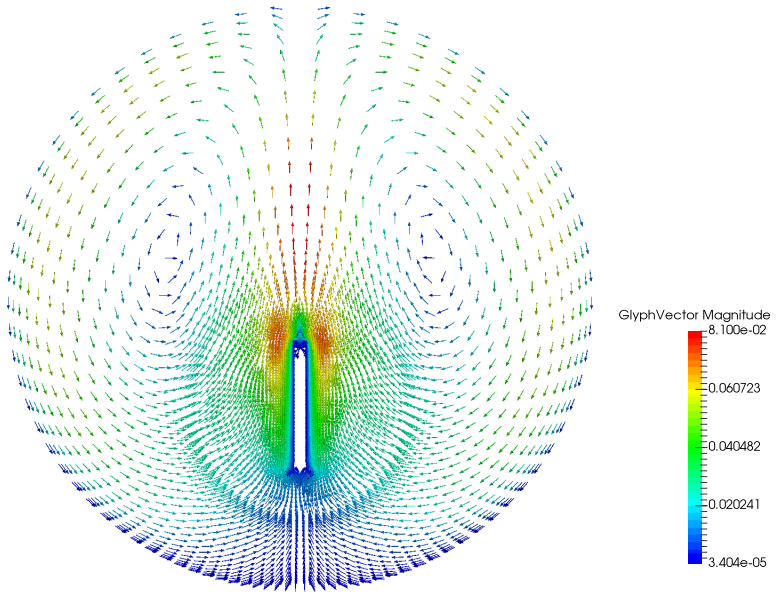

(b) $p_{rgh} = p - \rho_0(\vec{g} \cdot \vec{r})$

# Results

- Velocity Vector




(a) $p_{rgh} = p - \rho(\vec{g} \cdot \vec{r})$




(b) $p_{rgh} = p - \rho_0(\vec{g} \cdot \vec{r})$

# SIMPLE Algorithm

# Pressure-Velocity Coupling

- OpenFOAM's SIMPLE

$$\dot{m}_f = \rho_f \left\{ \frac{H(\vec{U}^*)}{a} \right\}_f \cdot \vec{S}_f \underbrace{-\rho_f \left( \frac{V}{a} \right)_f (\vec{g} \cdot \vec{r})_f \left( \frac{\partial \rho}{\partial n} \right)_f |\vec{S}_f|}_{\phi_g} - \rho_f \left( \frac{V}{a} \right)_f \left( \frac{\partial p}{\partial n} \right)_f |\vec{S}_f|$$

- Original SIMPLE(Rhie-Chow Interpolation)

$$\dot{m}_f = \underbrace{\rho_f \vec{U}_f^* \cdot \vec{S}_f - \rho_f \left( \frac{V}{a} \right)_f |\vec{S}_f| \left\{ \left( \frac{\partial p^*}{\partial n} \right)_f - \vec{n} \cdot \overline{(\nabla p^*)_f} \right\}}_{\text{Rhie-Chow interpolation}} - \rho_f \left( \frac{V}{a} \right)_f \left( \frac{\partial p'}{\partial n} \right)_f |\vec{S}_f|$$

$$= \rho_f \vec{U}_f^* \cdot \vec{S}_f + \rho_f \left( \frac{V}{a} \right)_f |\vec{S}_f| \vec{n} \cdot \overline{(\nabla p^*)_f} - \rho_f \left( \frac{V}{a} \right)_f \left( \frac{\partial p}{\partial n} \right)_f |\vec{S}_f|$$

  - $\overline{(\nabla p^*)_f}$ is interpolation of cell gradient

# Pressure-Velocity Coupling

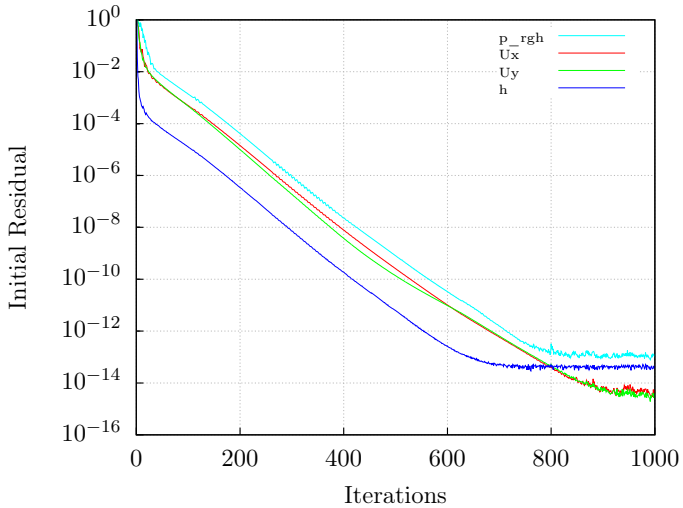
- $\phi_g$
  - Original SIMPLE algorithm의 $\dot{m}_f$는 $\phi_g$를 내포하고 있으므로 경계면에서만 $\phi_g$를 고려한다.

**pEqn.H**

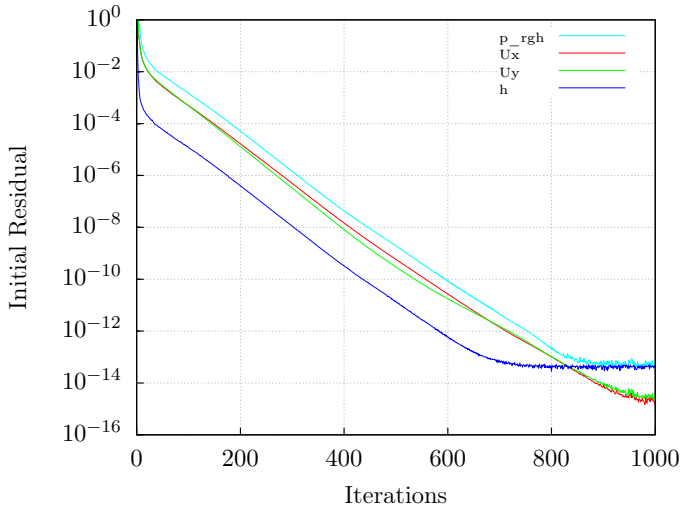
```
.
.
.
// Add flux due to gravity on all fixedFluxPressure boundaries
forAll(p_rgh.boundaryField(), patchi)
{
    if (p_rgh.boundaryField()[patchi].type() == "fixedFluxPressure")
    {
        phi.boundaryField()[patchi] += phig.boundaryField()[patchi];
    }
}

// Update the fixedFluxPressure BCs to ensure flux consistency
setSnGrad<fixedFluxPressureFvPatchScalarField>
(
    p_rgh.boundaryField(),
    (
       phi.boundaryField()
     - fvOptions.relative(mesh.Sf().boundaryField() & U.boundaryField())
      *rho.boundaryField()
    )/(mesh.magSf().boundaryField()*rhorAUf.boundaryField())
);

.
.
.
```

# Results

- Residual
  - $\rho_0$ : volume average



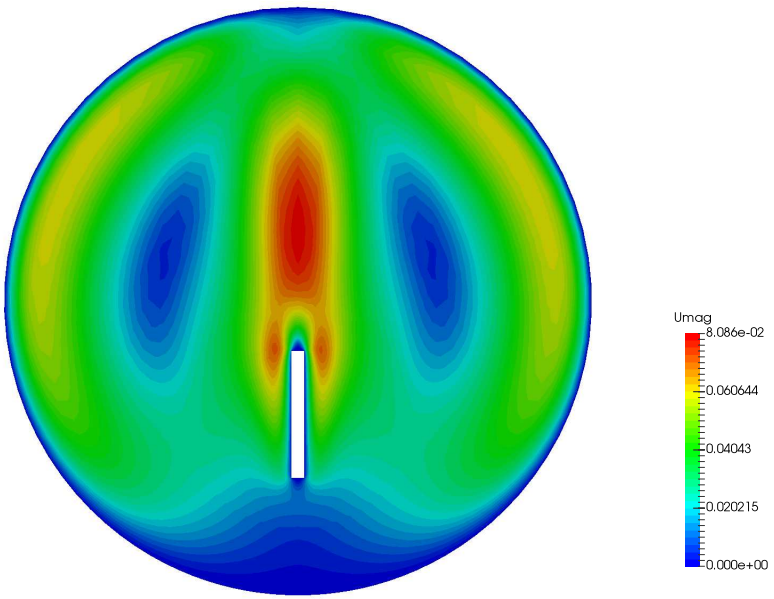

(a) OpenFOAM's SIMPLE



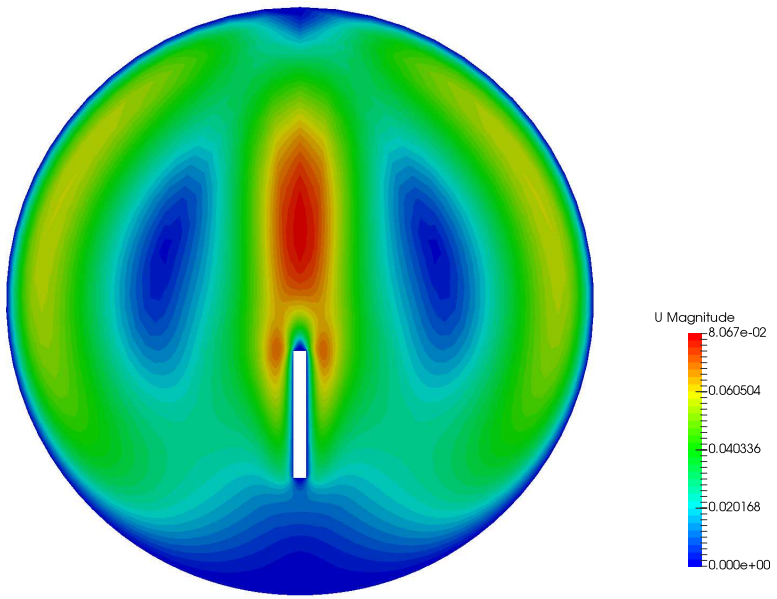

(b) Original SIMPLE

# Results

- Velocity Magnitude



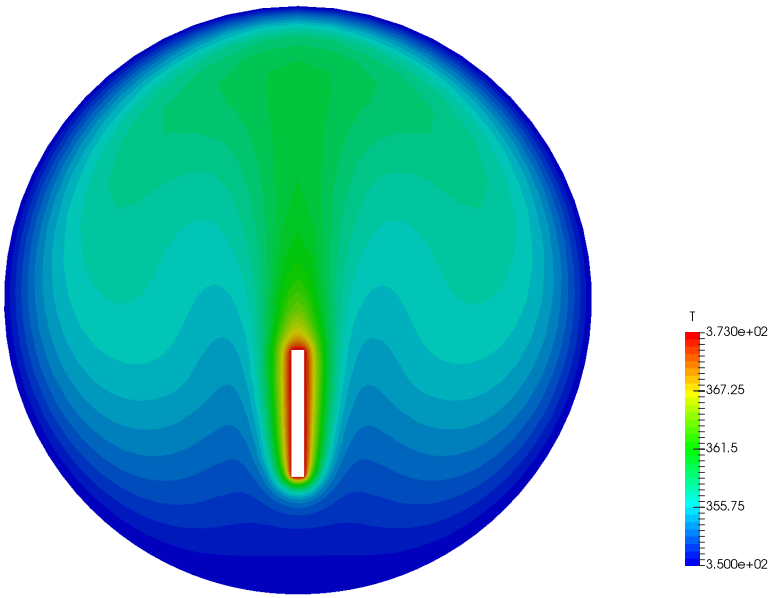

(a) OpenFOAM's SIMPLE



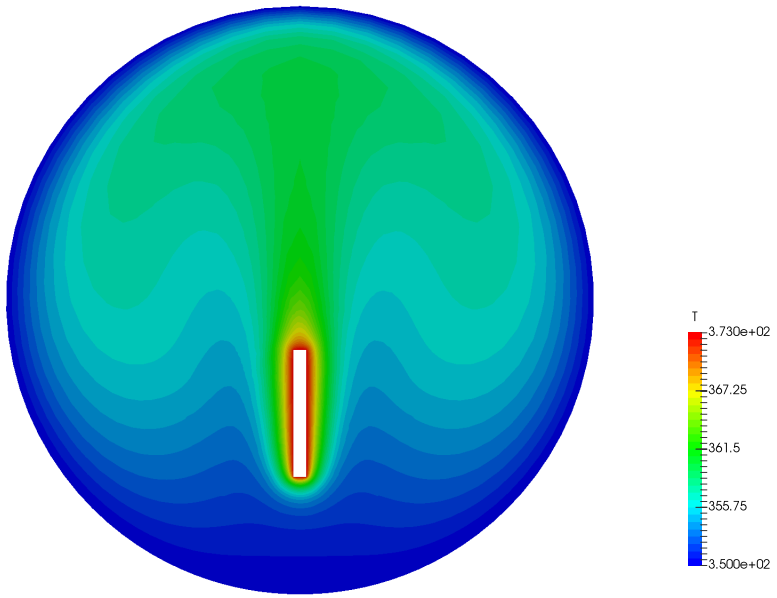

(b) Original SIMPLE

# Results

- Temperature



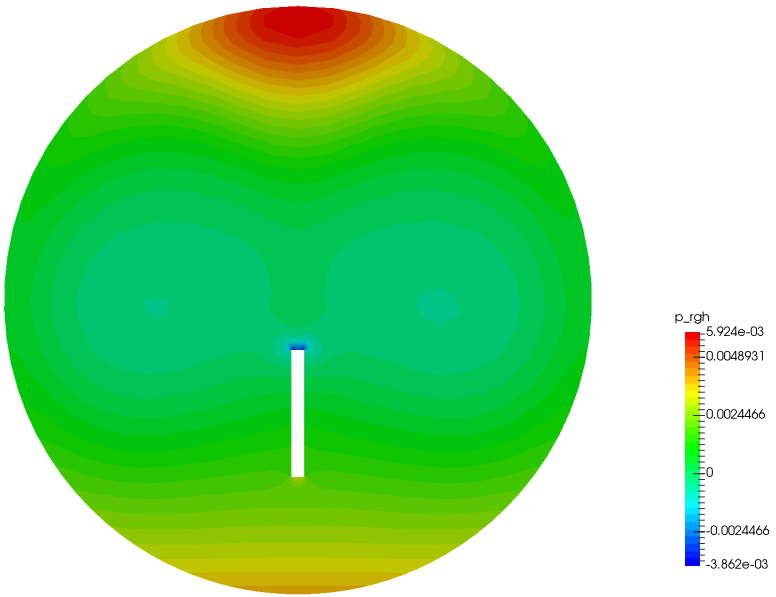

(a) OpenFOAM's SIMPLE



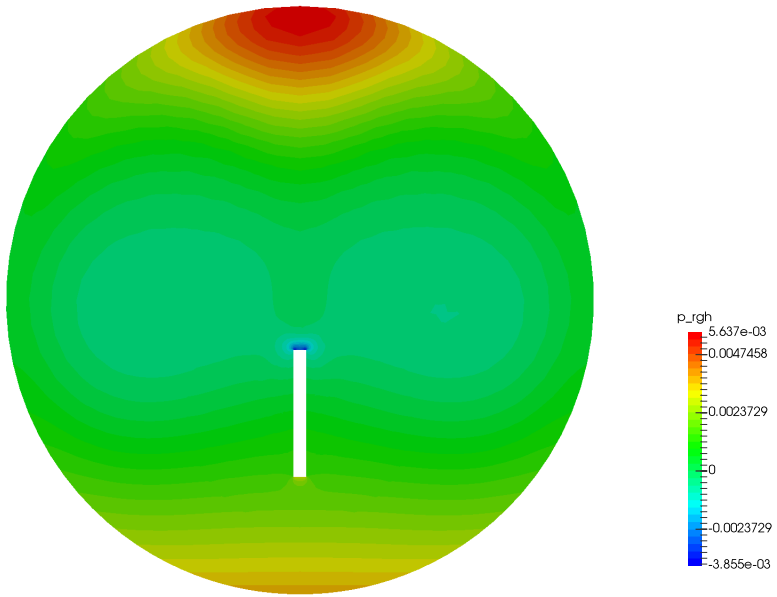

(b) Original SIMPLE

# Results

- Pressure($p\_rgh$)



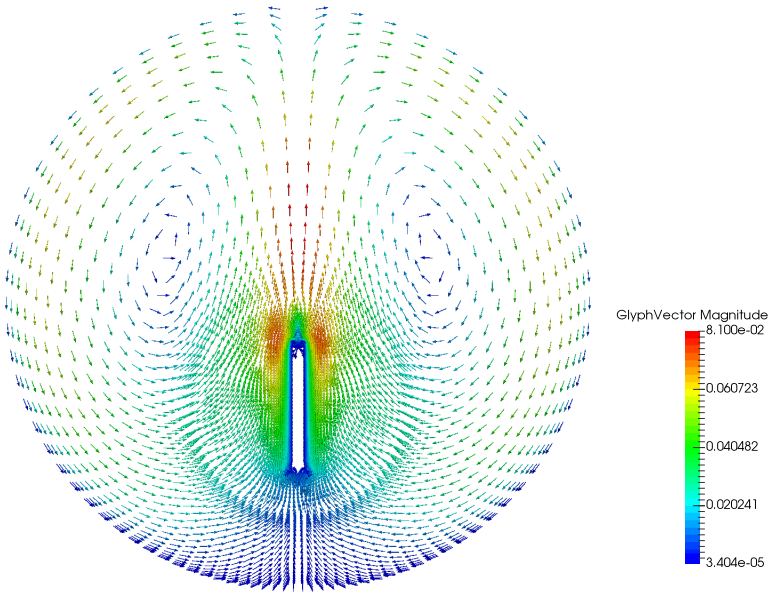

(a) OpenFOAM's SIMPLE



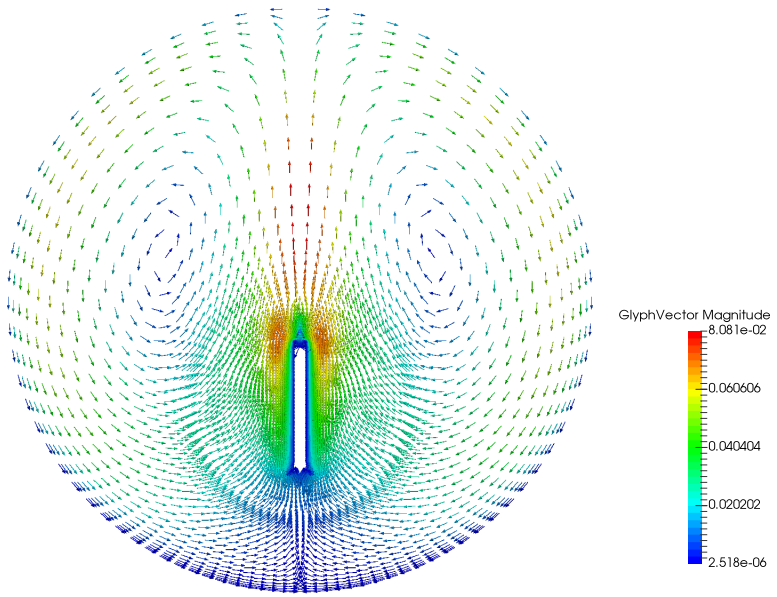

(b) Original SIMPLE

# Results

- Velocity Vector



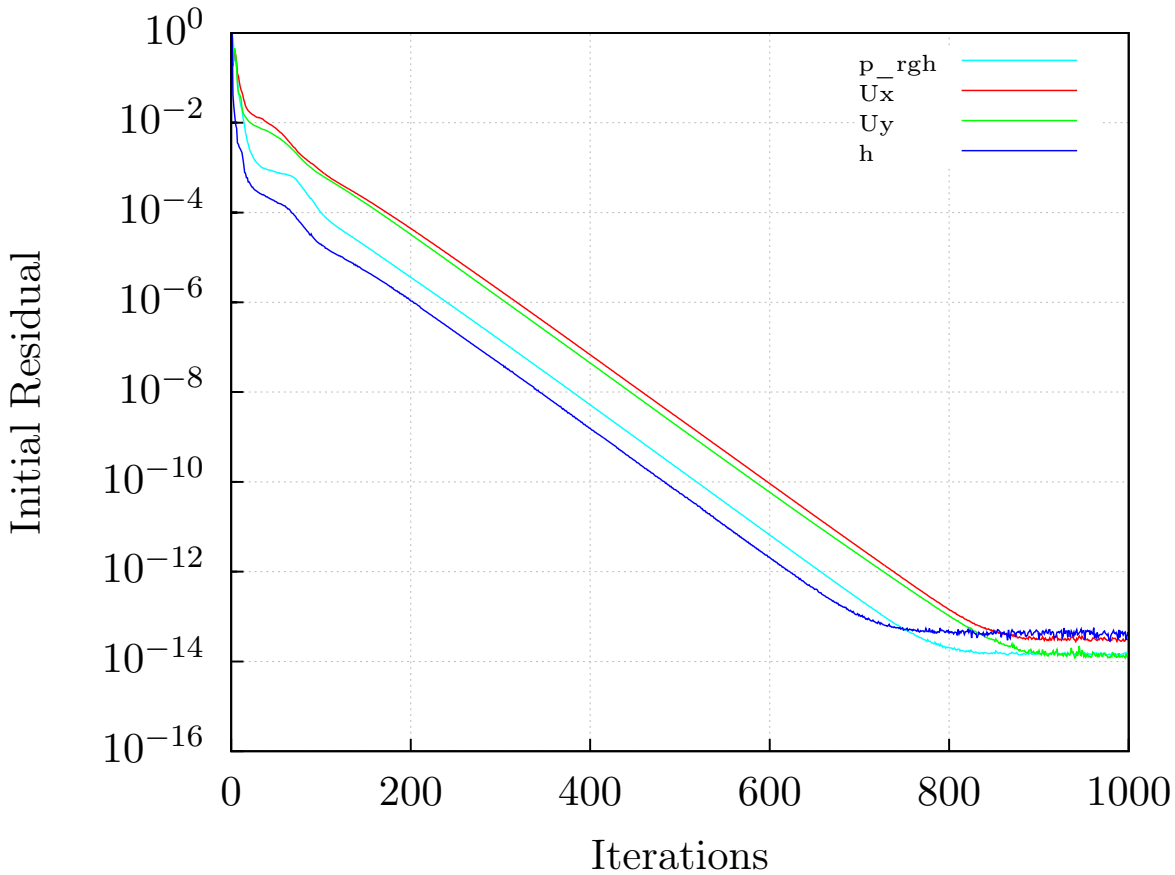

(a) OpenFOAM's SIMPLE




(b) Original SIMPLE

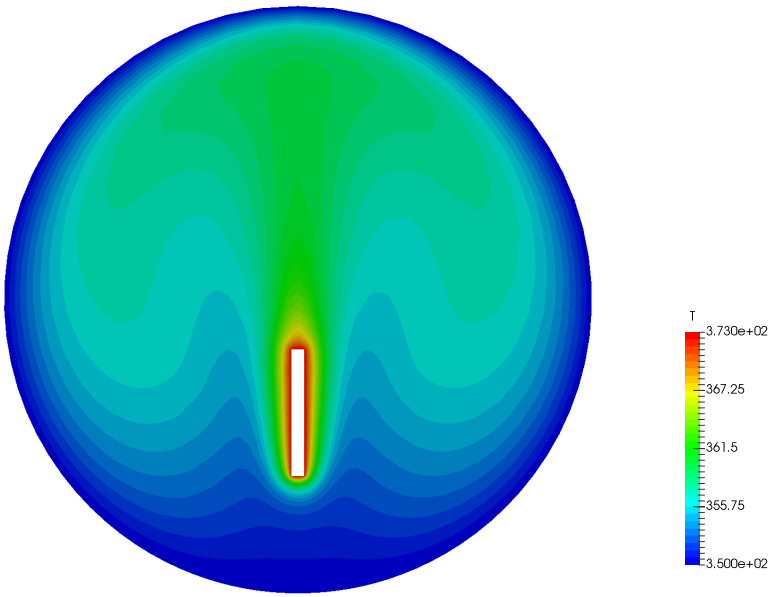
# What if we specify operating density($\rho_0$) as 0?

- Residual
  - $\rho_0 = 0$

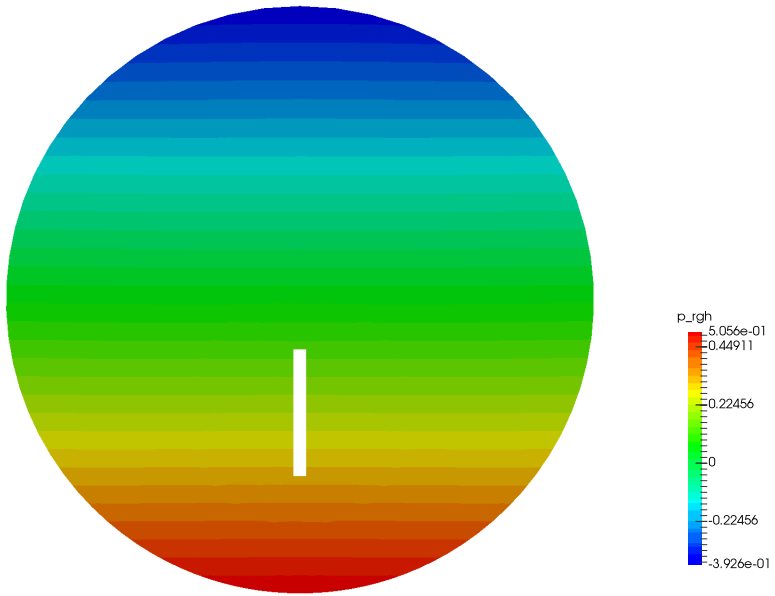


# What if we specify operating density($\rho_0$) as 0?

- Results



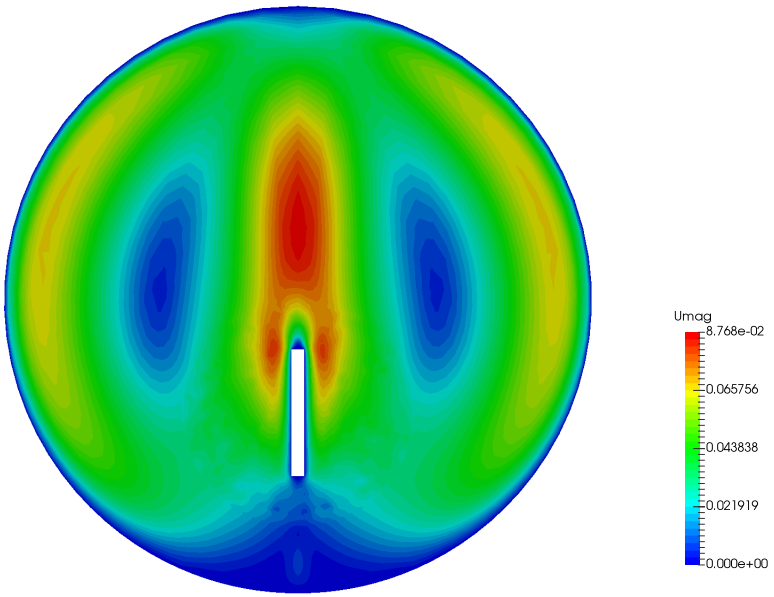

(a) Temperature



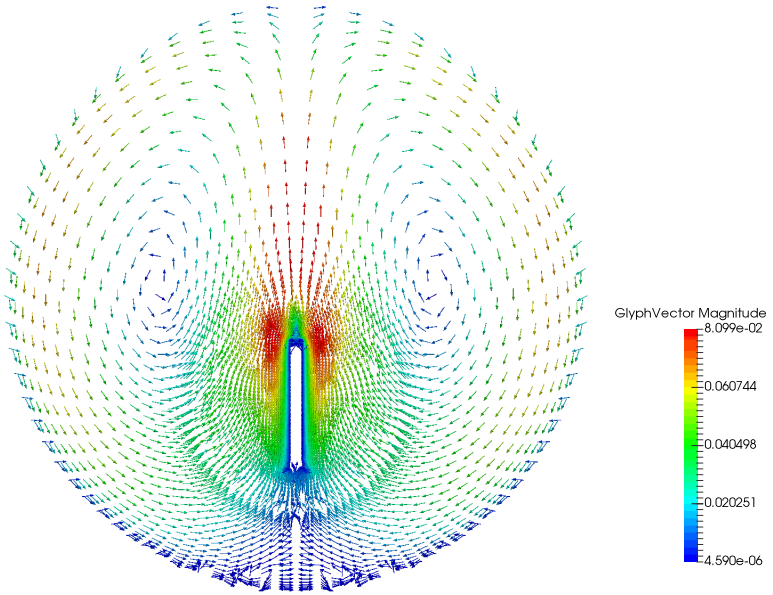

(b) Pressure(p_rgh)

# What if we specify operating density($\rho_0$) as 0?

- Results



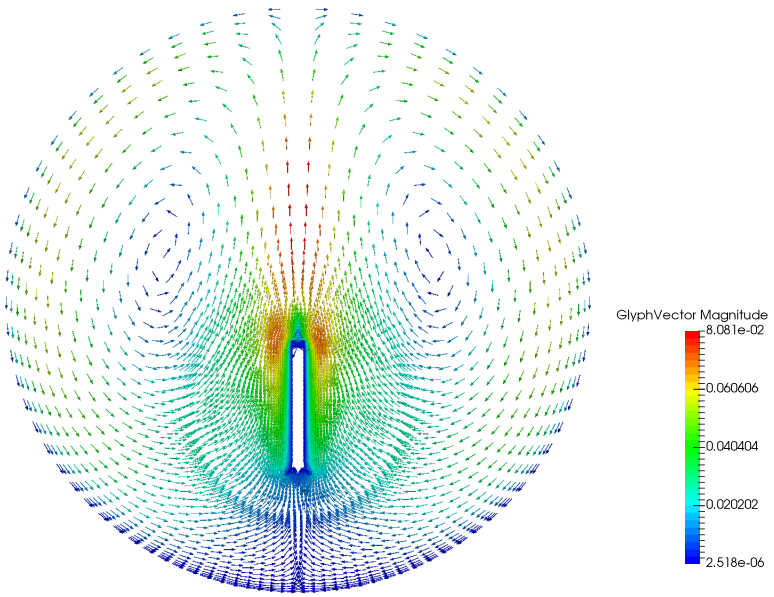

(a) Velocity Magnitude



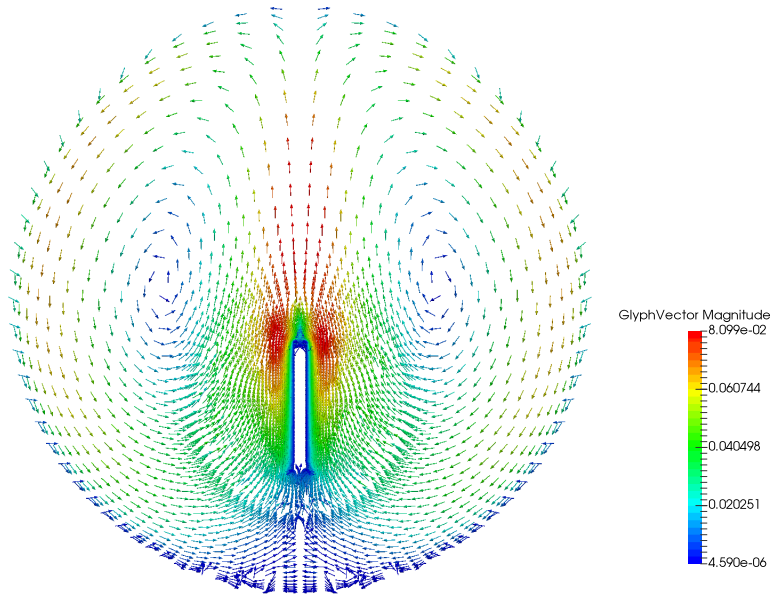

(b) Velocity Vector

# What if we specify operating density($\rho_0$) as 0?

- Velocity Vector




(a) $\rho_0$ = volume average




(b) $\rho_0 = 0$

# Pressure Gradient in Momentum Equation

# Discretization of Pressure Gradient in Momentum Equation

- Momentum Equation

$$\nabla \cdot (\rho \vec{U} \vec{U}) = \nabla \cdot \bar{\bar{\tau}} - \nabla p_{rgh} + (\rho - \rho_0)\vec{g}$$

- $\rho_0 = 0$ 이면 $p_{rgh} = p$

$$\boldsymbol{\nabla \cdot (\rho \vec{U} \vec{U}) = \nabla \cdot \bar{\bar{\tau}} - \nabla p + \rho \vec{g}} \tag{3}$$

- Integral Form

$$\int_{V_P} \left\{ \boldsymbol{\nabla \cdot (\rho \vec{U} \vec{U})} \right\} dV = \int_{V_P} (\boldsymbol{\nabla \cdot \bar{\bar{\tau}}}) dV - \int_{V_P} \boldsymbol{(\nabla p) dV} + \int_{V_P} (\boldsymbol{\rho \vec{g}}) dV \tag{4}$$

# Discretization of Pressure Gradient in Momentum Equation

- Two Different Discretization Methods in OpenFOAM (`fvc::grad(p)`)
  - Gauss' Divergence Theorem
    - flat face 가정
    - face centroid에 대한 Taylor series expansion 포함(first-order approximation)
    - face centroid에서의 pressure,$p_f$,는 interpolation으로 구함

$$\int_{V_P} (\nabla p) dV = \int_{V_P} (\nabla \cdot p\bar{\bar{I}}) dV = \oint_{\partial V_P} d\vec{S} \cdot (p\bar{\bar{I}}) \approx \sum_f (p\bar{\bar{I}})_f \cdot \vec{S}_f = \sum_f p_f \vec{S}_f \tag{5}$$

  - Least-Squares Method
    - $\nabla p$를 cell centroid에 대한 Taylor series expansion으로 표현
    - first-order approximation
    - cell centroid의 정의에 따라 1차항 소거
    - cell centroid에서의 pressure gradient, $(\nabla p)_P$,를 least-squares method로 구함

$$\int_{V_P} (\nabla p) dV \approx V_P (\nabla p)_P \tag{6}$$

# Discretization of Pressure Gradient in Momentum Equation

- `fvSchemes` dictionary example
  - Gauss' Divergence Theorem

**system/fvSchemes**

```
gradSchemes
{
    grad(p)         Gauss linear;
}
```

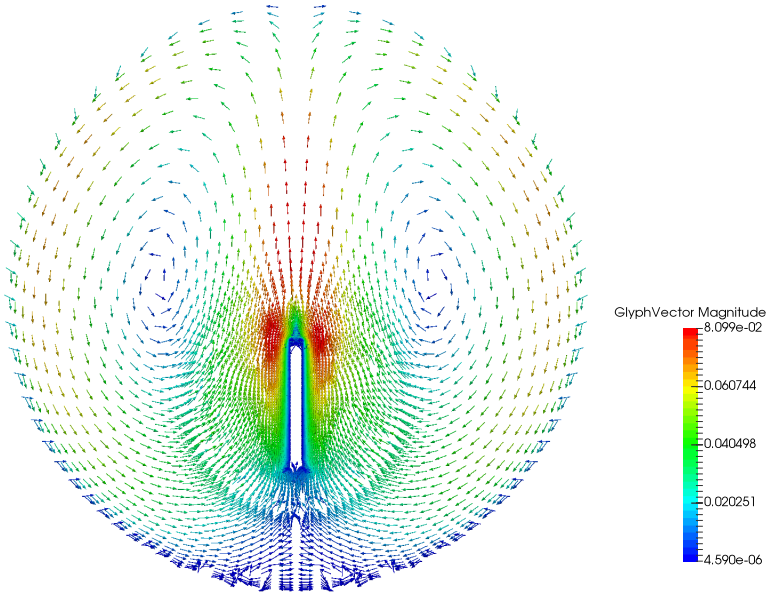
  - Least-Squares Method

**system/fvSchemes**

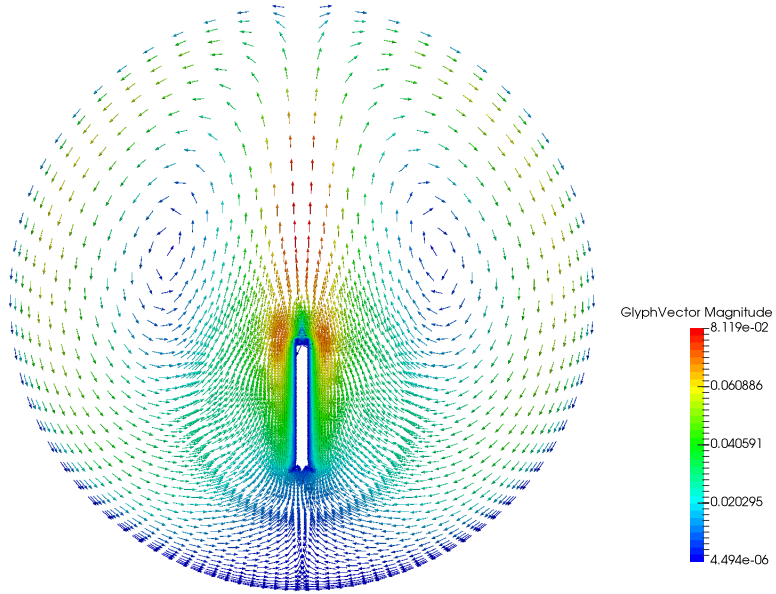
```
gradSchemes
{
    grad(p)         leastSquares;
}
```

# Discretization of Pressure Gradient in Momentum Equation

- Least-Squares Method
  - 벡터의 방향이 상당부분 개선됨



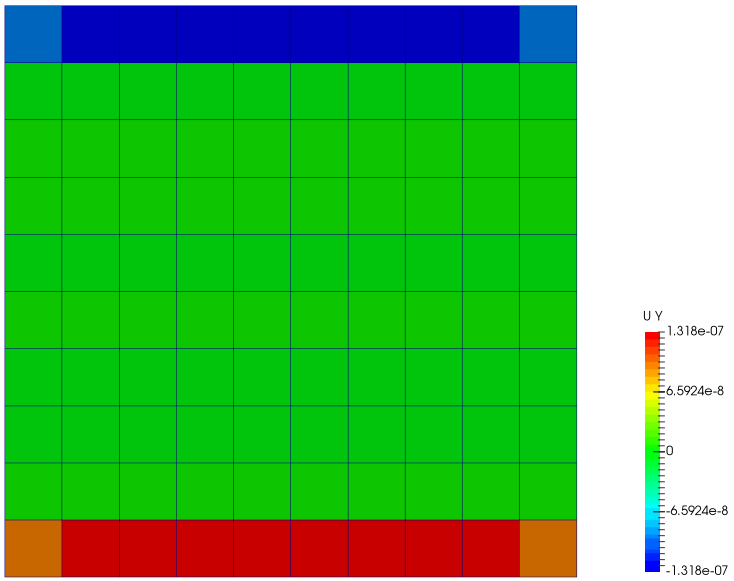

(a) `Gauss linear`



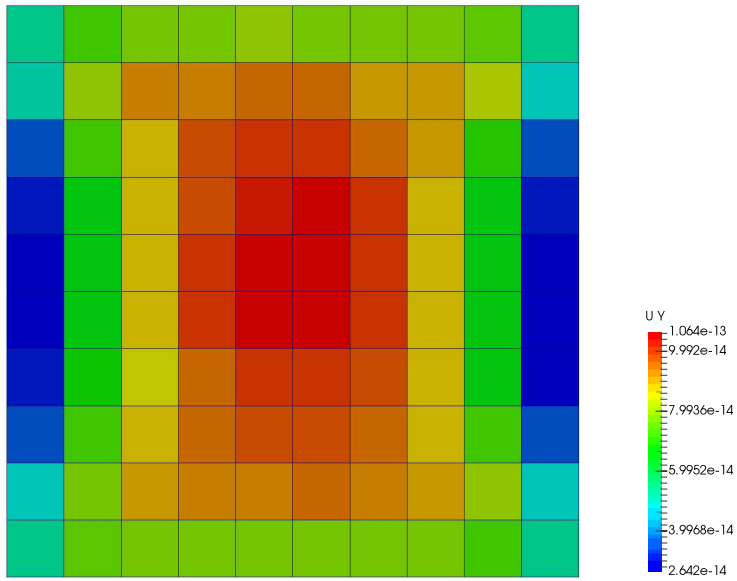

(b) `leastSquares`

# Discretization of Pressure Gradient in Momentum Equation

- Least-Squares Method의 문제점
  - 벽면에서 첫번째 cell의 gradient 계산 오류
  - `fixedFluxPressure` 경계조건과 부합하지 않음




(a) `leastSquares`



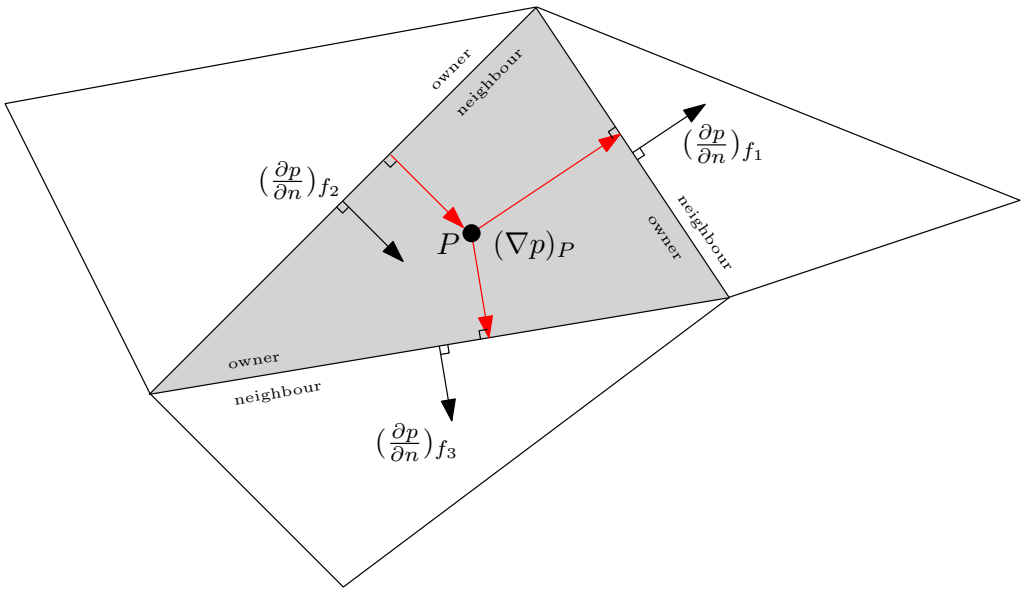

(b) `Gauss linear`

# Another Way of Pressure Gradient Calculation in OpenFOAM

- `fvc::reconstruct(fvc::snGrad(p)*mesh.magSf())`

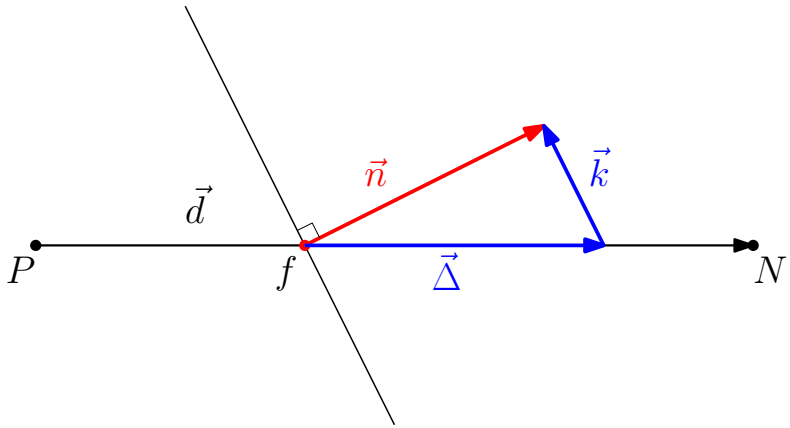
$$\int_{V_P} (\nabla p) dV \approx V_P (\nabla p)_P$$

$$\left\{ \sum_f \rho_f \left( \frac{V}{a} \right)_f (\vec{n} \otimes \vec{S}_f) \right\} \cdot (\nabla p)_P = \sum_f \rho_f \left( \frac{V}{a} \right)_f \left( \frac{\partial p}{\partial n} \right)_f \vec{S}_f$$



# Another Way of Pressure Gradient Calculation in OpenFOAM

- `fvc::snGrad(p)`
  - non-orthogonal correction




- Split unit normal vector($\vec{n}$) into two parts:

$$\vec{n} = \vec{\Delta} + \vec{k}$$

then,

$$\left(\frac{\partial p}{\partial n}\right)_f = |\vec{\Delta}|\frac{p_N - p_P}{|\vec{d}|} + \underbrace{\vec{k} \cdot (\nabla p)_f}_{\text{non-orthogonal correction}}$$

# Another Way of Pressure Gradient Calculation in OpenFOAM

- `fvc::snGrad(p)`
  - correction을 위해 필요한 cell gradient 처리 방법 수정 필요

**correctedSnGrad.C**

```
    .
    .
    .
Foam::fv::correctedSnGrad<Type>::fullGradCorrection
(
    const GeometricField<Type, fvPatchField, volMesh>& vf
) const
{
    const fvMesh& mesh = this->mesh();

    // construct GeometricField<Type, fvsPatchField, surfaceMesh>
    tmp<GeometricField<Type, fvsPatchField, surfaceMesh> > tssf =
        mesh.nonOrthCorrectionVectors()
      & linear<typename outerProduct<vector, Type>::type>(mesh).interpolate
        (
            gradScheme<Type>::New
            (
                mesh,
                mesh.gradScheme("grad(" + vf.name() + ')')
            )().grad(vf, "grad(" + vf.name() + ')')
        );
    tssf().rename("snGradCorr(" + vf.name() + ')');

    return tssf;
}

    .
    .
    .
```

# Another Way of Pressure Gradient Calculation in OpenFOAM

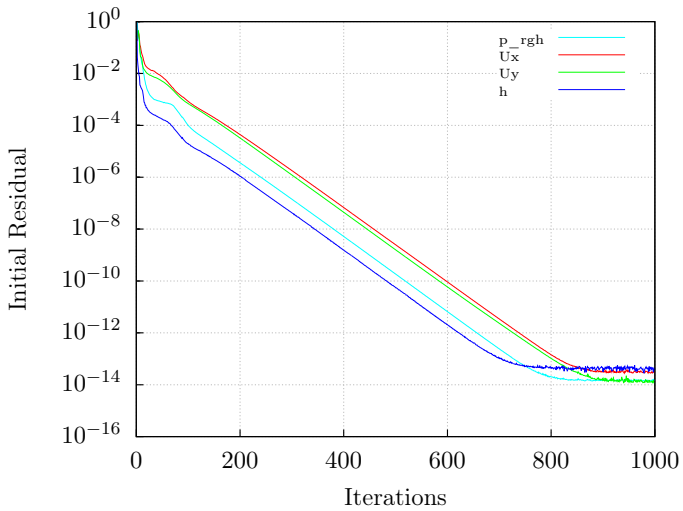
- `fvc::snGrad(p)`
  - objectRegistry에 등록된 gradient field를 찾아서 사용하도록 수정

**correctedSnGrad.C**

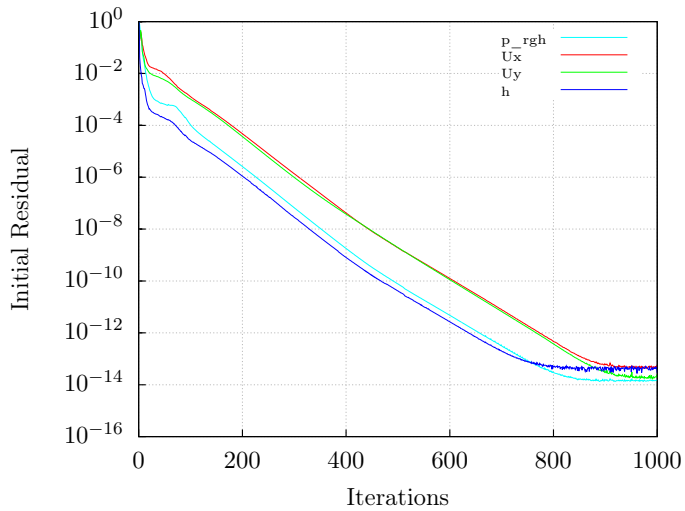
```
.
.
.
tmp
<
    GeometricField
    <
        typename outerProduct<vector, Type>::type,
        fvPatchField,
        volMesh
    >
> tgradVf
(
    mesh.lookupObject
    <
        GeometricField
        <
            typename outerProduct<vector, Type>::type,
            fvPatchField,
            volMesh
        >
    >("grad" + vf.name())
);

.
.
.
```

# Results

- Residual
  - $\rho_0 = 0$



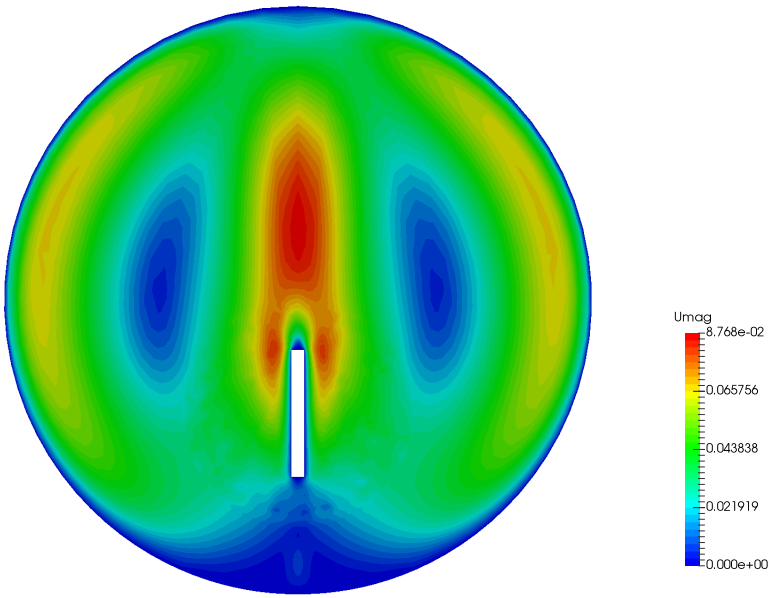

(a) Gauss linear



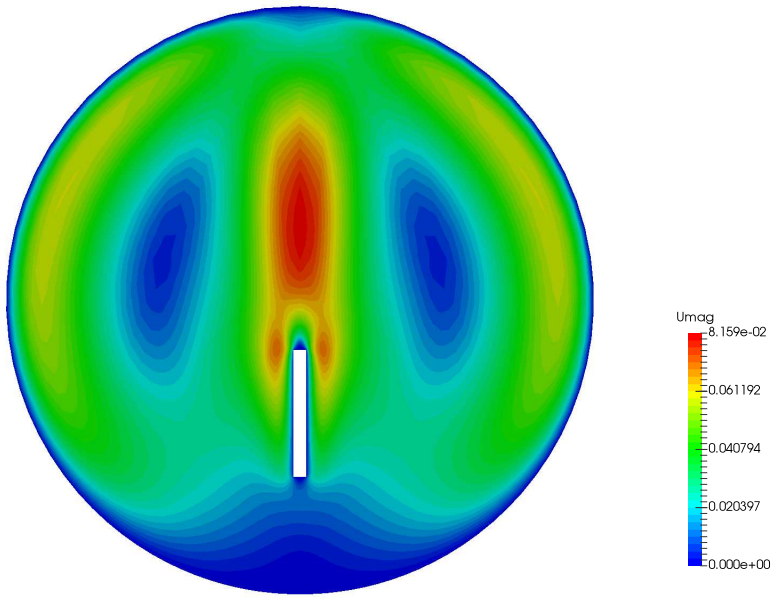

(b) reconstruct snGrad

# Results

- Velocity



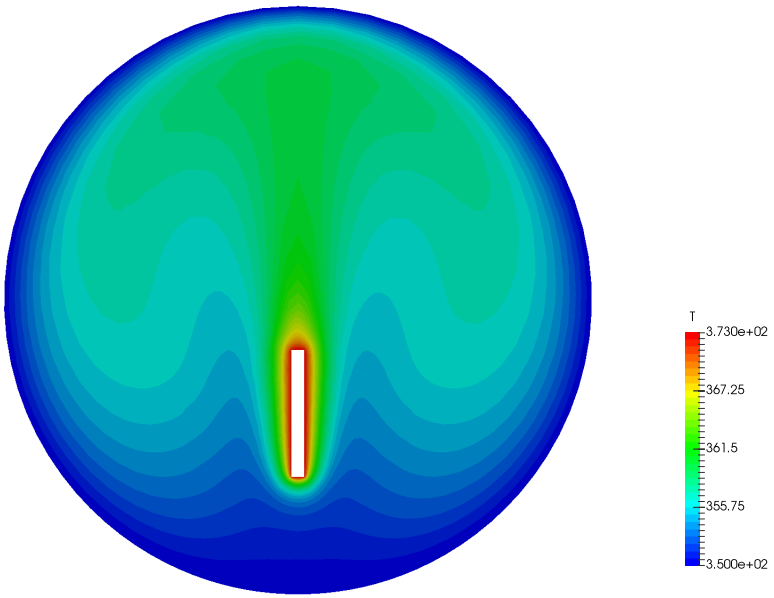

(a) Gauss linear



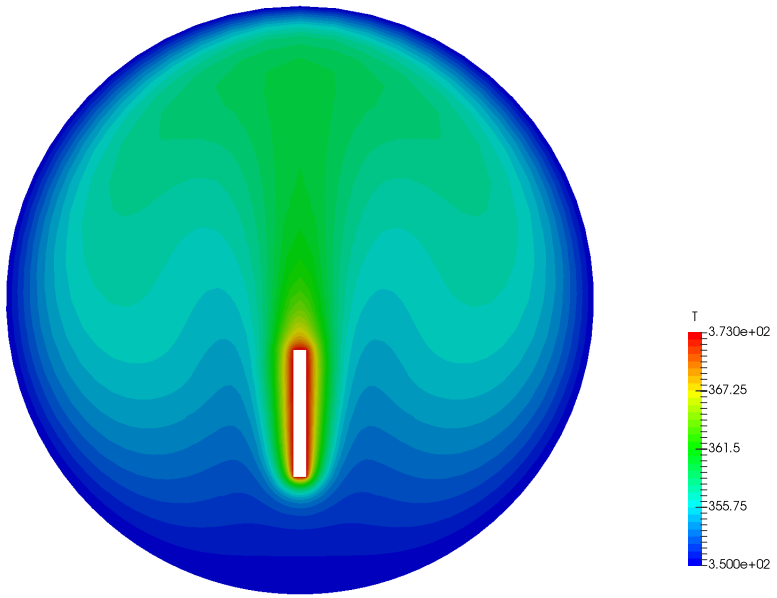

(b) reconstruct snGrad

# Results

- Temperature



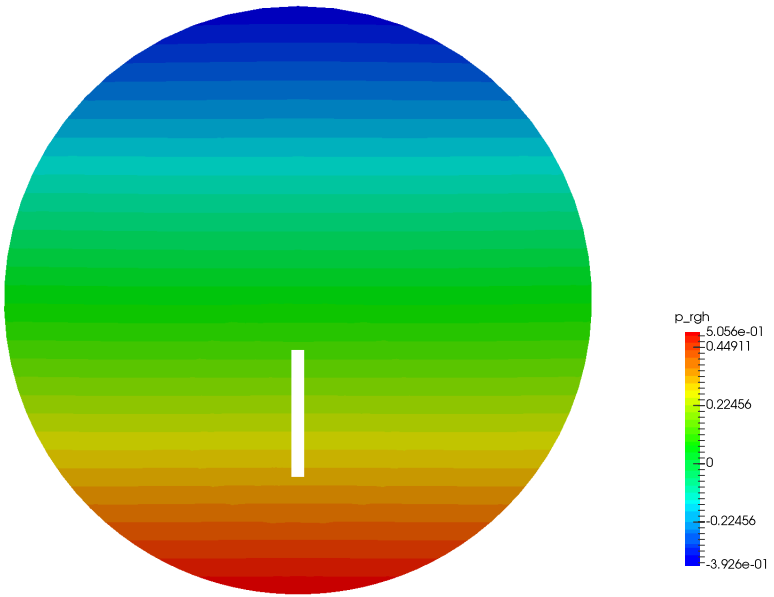

(a) Gauss linear



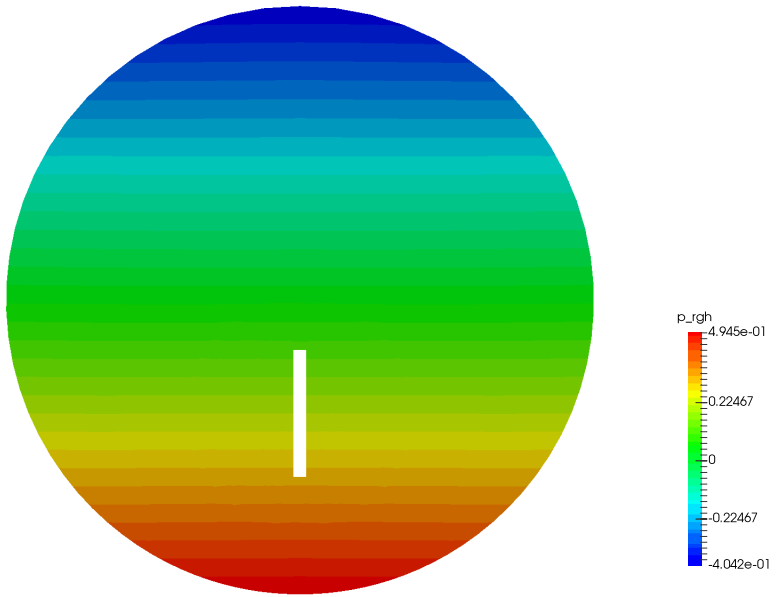

(b) reconstruct snGrad

# Results

- Pressure($p\_rgh$)



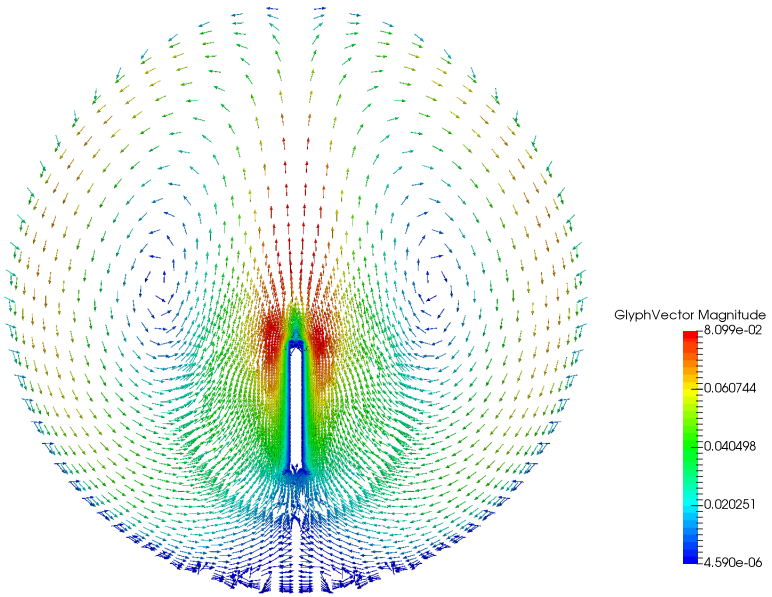

(a) Gauss linear



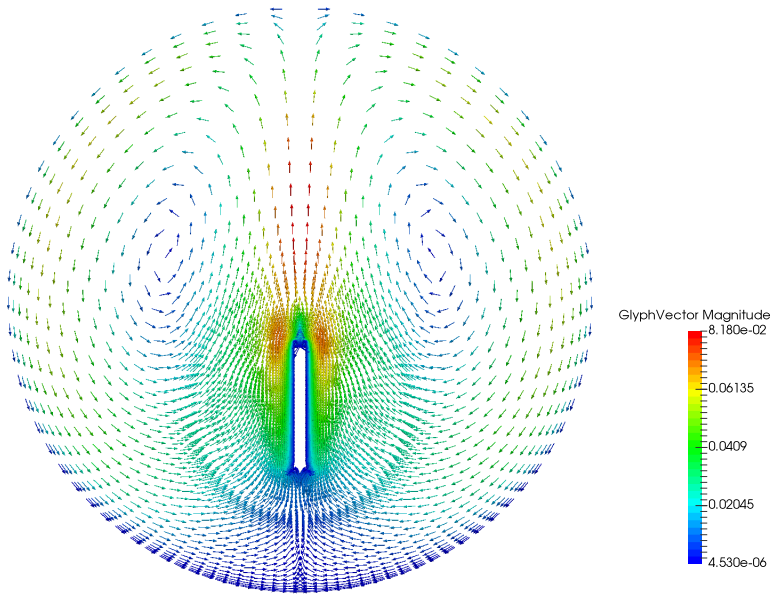

(b) reconstruct snGrad

# Results

- Velocity Vector



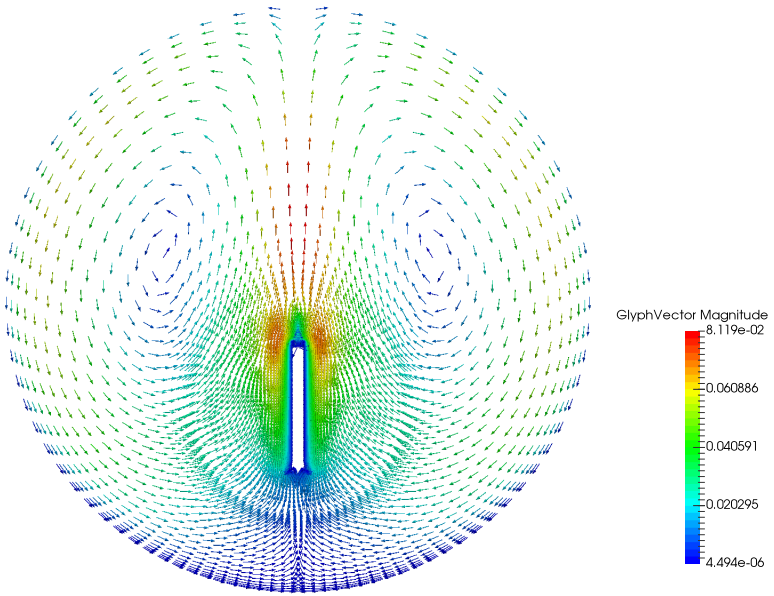

(a) Gauss linear



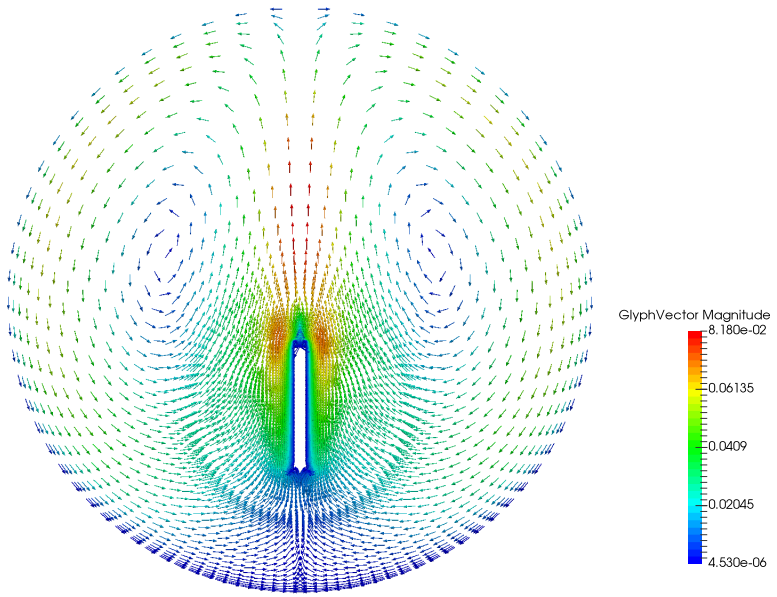

(b) reconstruct snGrad

# Results

- Velocity Vector




(a) leastSquares




(b) reconstruct snGrad

# 부록

## Limited Gradient 사용의 문제점

# Limited Gradient 사용의 문제점

- `cellLimited` gradient scheme
  - dictionary

**system/fvSchemes**

```
gradSchemes
{
    grad(p)         cellLimited Gauss linear 1.0;
}
```

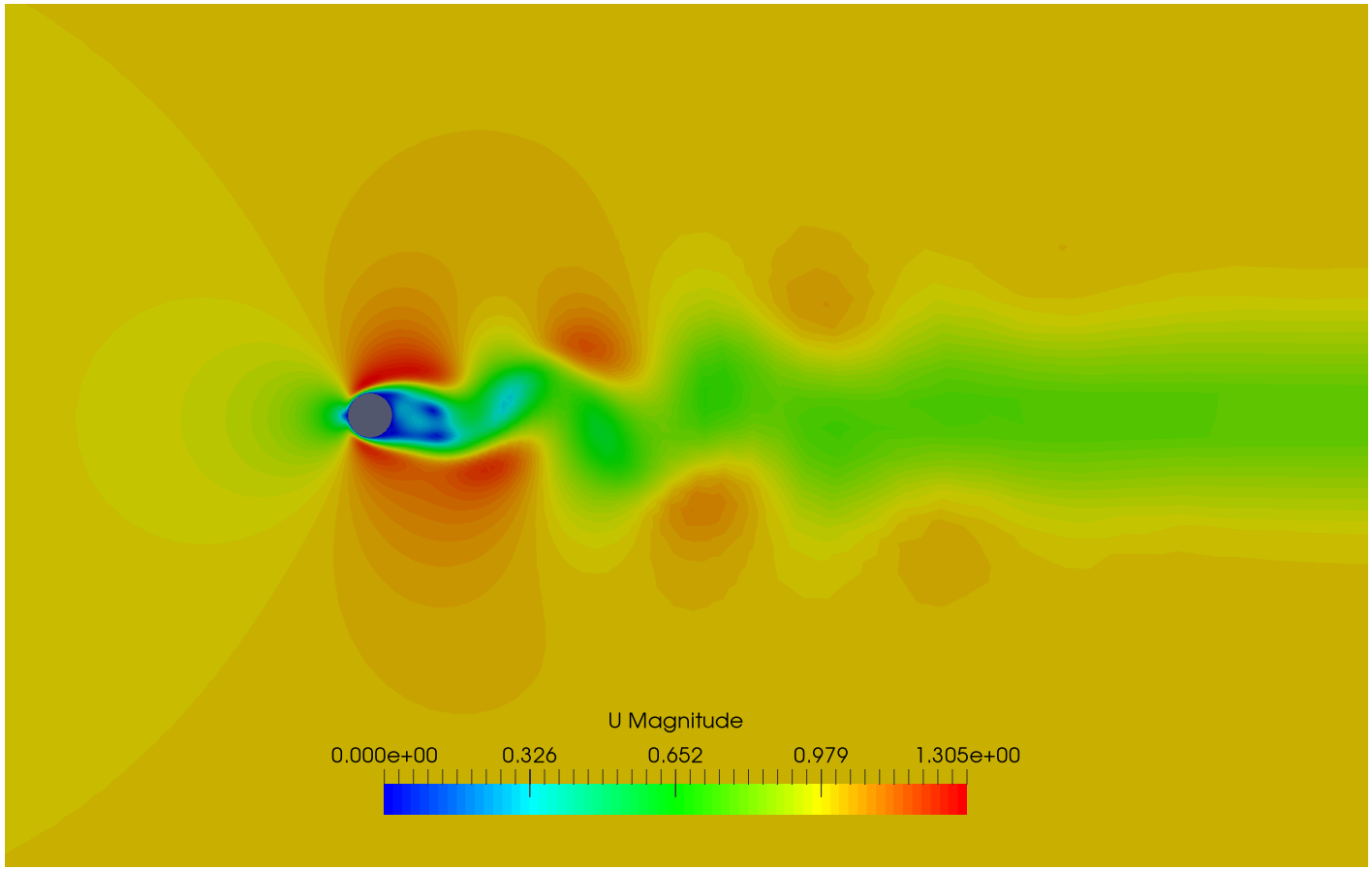
- `solve(UEqn() == -fvc::grad(p));` 의 의미

$$\int_{V_P} \left\{ \nabla \cdot (\rho \vec{U} \vec{U}) \right\} dV = \int_{V_P} (\nabla \cdot \bar{\bar{\tau}}) dV - \underbrace{\Psi \int_{V_P} (\nabla p) dV}_{???} + \int_{V_P} (\rho \vec{g}) dV$$

$\Psi$ : slope limiter

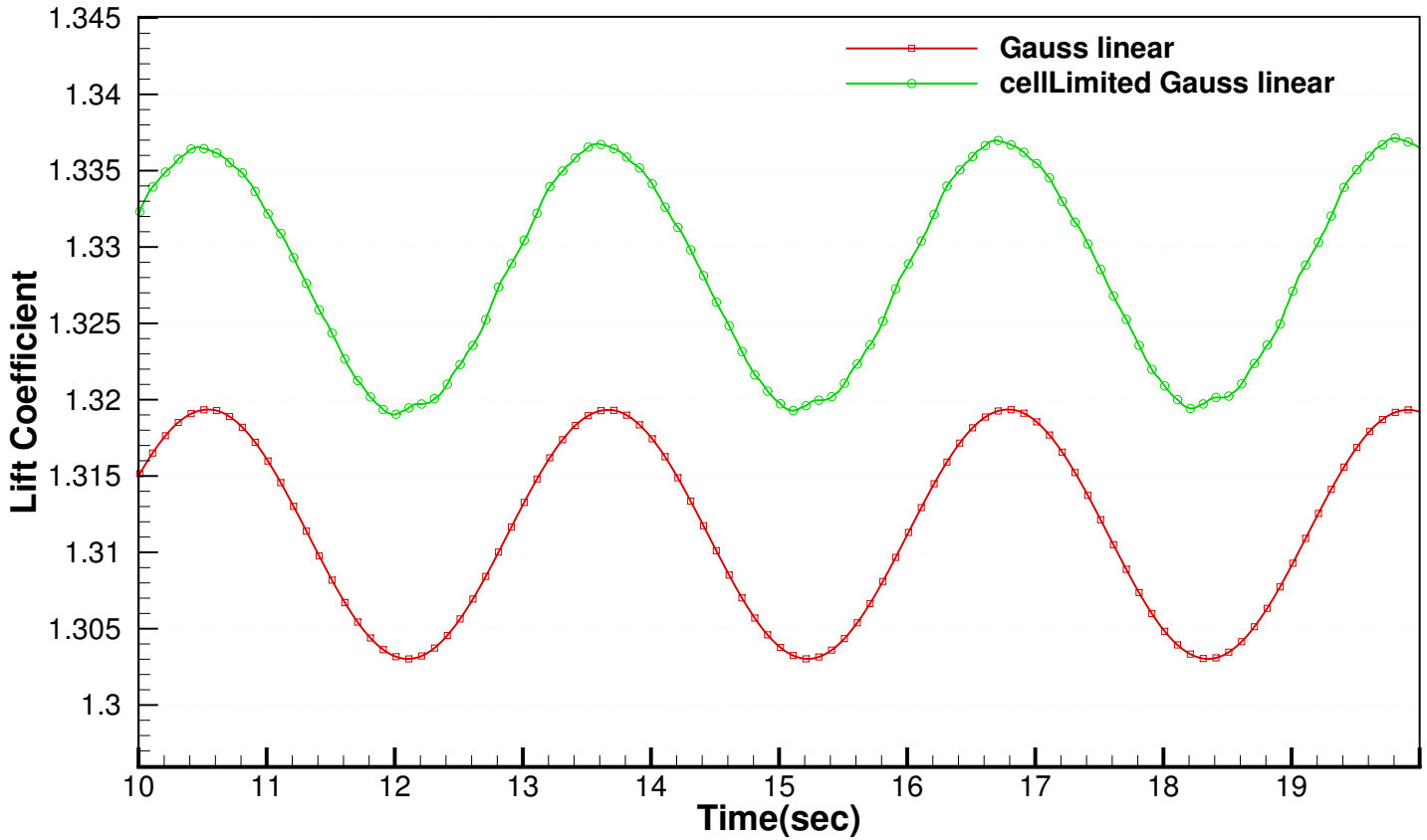
# Limited Gradient 사용의 문제점

- laminar flow past a circular cylinder



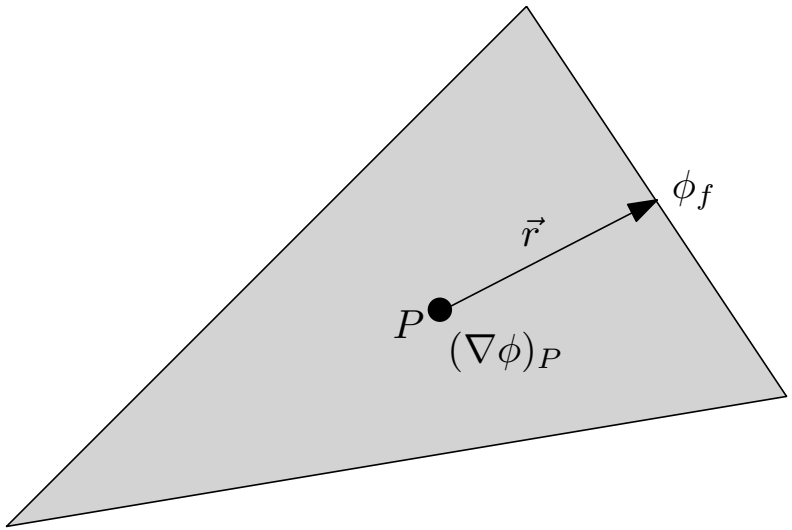
# Limited Gradient 사용의 문제점

- Lift Coefficient



# Slope Limiter

- Face value를 reconstruct할 때 사용



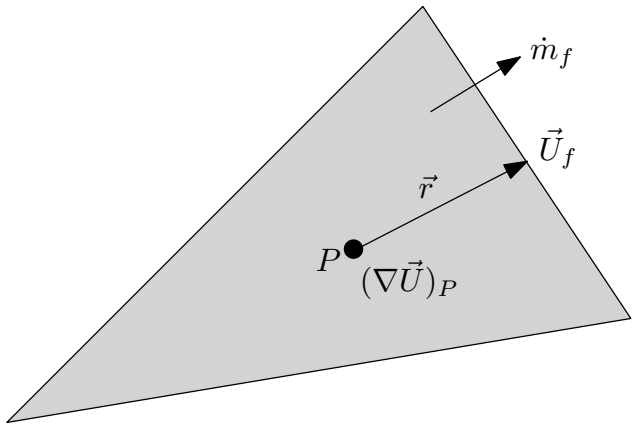

$$\phi_f = \phi_P + \vec{r} \cdot \{\Psi(\nabla\phi)_P\}$$

# Slope Limiter

- Convection Term 이산화

$$\int_{V_P} \left\{ \nabla \cdot (\rho \vec{U} \vec{U}) \right\} dV \approx \sum_f \dot{m}_f \vec{U}_f$$

  - `linearUpwind`




$$\vec{U}_f = \vec{U}_P + \vec{r} \cdot \{ \Psi (\nabla \vec{U})_P \}$$

# Slope Limiter

- `fvSchemes` dictionary

**system/fvSchemes**

```
gradSchemes
{
    default         Gauss linear;
    grad(U)         cellLimited Gauss linear 1.0;
}

divSchemes
{
    default          Gauss upwind;
    div(phi,U)       Gauss linearUpwind grad(U);

    .
    .
    .

    div((muEff*dev2(T(grad(U))))) Gauss linear;
}
```