OpenFOAM 覚書

Apr. 14, 2025, 更新

1. 資料

(1) OpenFOAM Foundation
https://openfoam.org/
ver7 が最新（2020April 現在）.

(2) OpenCFD
https://www.openfoam.com/
v2006 が最新（2020April 現在）. 我々が使っているやつ.
v2012 が最新（2021April 現在）.
v2106 が最新（2021Oct.現在）. ここから，ファイルフォーマットが変更になり，菊池研ライブラリが使えなくなる.

(3) Penguine
http://penguinitis.g1.xrea.com/study/OpenFOAM/
http://penguinitis.g1.xrea.com/index.html

(4) 言語：C++入門
http://wisdom.sakura.ne.jp/programming/cpp/
とても分かりやすいので，クラスの概念の分からない人はここから学ぶ.

2. 型

基本型
- label 整数（int or long int?）
- scalar スカラー（実数: float or double）
- vector ベクトル（3 or 6 のみか？）
- word 文字列
- bool ブーリアン

```
Tensor<scalar> t(
        1., 2., 3.,
        4., 5., 6.,
        7., 8., 9.
);      // テンソル　t.xx()でアクセス.
SymmTensor<scalar> t2 = symm(t);
Tensor<scalar> t2 = skew(t); // 1./2.*(t - t.T())
```

フィールド型
scalarField, labelField
Field<****>と****Field の違いは，サイズ定数があるかないか.
ex.

```
Field<scalar>       A;
ScalarField         B;
```

の場合，B.size()が使える.

```
for(int i=0; i<N; i++ )         A[i] = ***;
forAll( B, i )          B[i] = ***;
```

point　ベクトル，重心位置などに使っている.

autoPtr　自動で開放してくれるポインタ型

単位付き型
- dimensionedScalar スカラー
- dimensionedVector ベクトル

基本型リスト
- labelList
- scalarList
- vectorList
- wordList
- boolList

単位付き型リスト
List<dimensionedScalar> a;

- 行列型

https://www.openfoam.com/documentation/guides/latest/api/dir_b07477ab37bc0cd7eb28fb569c691c21.html
正方行列
#include "SquareMatrix.H"
SquareMatrix<scalar> A(行数)；

行列
対称正方行列
対角行列
いろいろ. C 記述の二重ポインタや三重ポインタは関数間での受け渡しに失敗する. 書式が違うようだがよく分からず. よって，いちいち定義された行列などを使うことになる.

絶対値
scalar a = mag(x);
（要注意：クラスにもよるかもしれないが，absf は使えず，abs は正しくない答えを返してくる．ex. abs(-0.1)=0）

累乗
scalar a;
a = sqr(x);　　// 2 乗
a = pow2(x);　 // 2 乗
a = pow3(x);　 // 3 乗
a = pow4(x);　 // 4 乗
a = pow5(x);　 // 5 乗
a = pow6(x);　 // 6 乗
a = pow025(x); // 0.25 乗
a = pow(x, r); // r 乗

動力学関係
Foam::RBD::restraints::externalForce:: rigidBodyModelState 型
member
scalarField q():　　7(重心ベクトル，姿勢ベクトル，あとひとつ)
scalarField qDot(): 7(上記の一回微分)
scalarField qDdot(): 7(上記の 2 回微分)

state.t():　　現在の時間
state.deltaT()　　現在の時間刻み

3. ベクトル計算

scalar k;
vector a, b, c;　で宣言
a[0], a[1], a[2]でアクセス可
a.x(), a.y(), a.z()も有.

c=a+b;　// ベクトル和
c=k*a;　//でスカラー倍
k = a & b; // 内積計算
c = a ^ b; // 外積

Tensor<scalar> A,B;とすると，A, B は 3 行×3 行の行列．内部は一次元配列．A=(1,2,3,4,5,6,7,8,9).
A & B で行列の掛け算
A.T()で A の転置
vector a; として，
a&A
A&a
でベクトルと行列の掛け算ができる．a が縦ベクトルなのか，横なのかは勝手に判断されるので，後述するように注意が必要．
剛体計算のプログラムグループは少なくとも横ベクトルを想定しているようだ．
A.xx(), A.xy()などで内部の成分にアクセスできる．
det(A)で，行列式の値を返す．A.det()で返さないところが不思議だ．

Info でベクトルを表示した場合，Zero ベクトルになると，スカラーのように 0 と表示される．例えば，6 成分ある加速度 qDdot は，ノルムが 0 でない場合には，

- 6(
  0
  0
  0
  5.1
  2
  0
)

のように表示されるが，ノルムが 0 になると，

- 6{0}

と表示される．{}は省略しているという意味か？

4. 定義定数

どのクラスかは調べていない．Foam::RBD::restraints::externalForce:: ?
vector Zero; たぶん Zero=(0,0,0);
nullptr は，ヌルポインタ．ポインタの中身が空宣言
bodyID_: ID of the body the restraint is applied to とのこと．関節の数をナンバリングしている．Pxyz も Px もひとつと数えている．rigidbody に対して，jointbody と呼んでいるようだ．rigidBodyRestraint.H に定義されている．以下も同じ．
bodyIndex_: Index of the body the force is applied to
spatialTensor::I　　たぶん，単位行列
VSMALL: 1e-300
constant::mathematical::pi　　π
なお，変数には_を付けるルールのようだ．A()は関数．A_は変数

5. 関数

```
scalar a;
vector v;
Tensor<scalar> t ;

a = sqrt(x);      // 0.5 乗
a = sqr(x);       // 2 乗
a = pow2(x);      // 2 乗
a = pow3(x);      // 3 乗
a = pow4(x);      // 4 乗
a = pow5(x);      // 5 乗
a = pow6(x);      // 6 乗
a = pow025(x); // 0.25 乗
a = pow(x, r); // r 乗

a = det(t);// determinant
v = eigenValues(t);   // eingenvalue

scalar a = mag(t);      // ::sqrt(magSqr(t))
scalar b = magSqr(t); //  各成分の  magSqrt()  の和
```

Foam::atan2(y,x)として呼び出さないと正しい答えが返ってこない...　？？？？

rigidBodyDynamics 関係

. point bodyPoint(vector or point?);

重心位置ではなく，回転中心から引数分だけ並進したベクトルを返す．Zero を送ると，回転中心を返す．

return((model_.X0(bodyID_).inv() && spatialVector(Zero, p)).l())と定義されている．

(model_.X0(bodyID_).inv() && spatialVector(Zero, Zero)).l()は，以下に示す model_.X0(bodyID_).r()と同じ．

(model_.X0(bodyID_).inv()は，姿勢（逆行列か？）＋回転中心ベクトル．ex. (1 0 0 0 1 0 0 0 1) (0.075 -0 -0)

上記から想像すると，

model_.X0(bodyID_).inv() -> 姿勢行列 R（逆行列か？），回転中心ベクトル p,

spatialVector(a, b) -> a, b ベクトルとして，

演算子&&は，

p+Ra+b

のあと，6 成分（姿勢ベクトル，位置ベクトル）を返すという計算か？　**.l()は後ろ 3 成分を取り出している**.

vector spatialVector(vector, vector); 3 成分二つを組み合わせて 6 成分ベクトルとして返す？　ex. fx[bodyIndex_] += spatialVector(moment, force);

. spatialVector bodyPointVelocity(point)：**重心ではなく，回転中心から** p 点の位置における速度

model_.v(bodyID_, p)と定義されている．

model_.X0(bodyID_).E()	bodyID で示される剛体の姿勢行列を返す．**x, y, z 軸ベクトルが横に並べられている．ロボ定義に対して転置されている．**
model_.X0(bodyID_).r()	回転中心の基準座標系からの位置ベクトル
model_.v(bodyID_)	bodyID で示される剛体の回転中心の速度（角速度ベクトル，速度ベクトルの順）を返す．6 成

分
model_.v(bodyID_,p)　　bodyID で示される剛体の回転中心から p ベクトル離れた場所の速度（角速度ベクトル，速度ベクトルの順）を返す．6 成分
model_.a(bodyID_)　　bodyID で示される剛体の（回転中心の）加速度（角速度ベクトル，速度ベクトルの順）を返す．6 成分
model_.v(bodyID_).w()　　bodyID で示される剛体の角速度ベクトルを返す．3 成分
model_.v(bodyID_).l()　　bodyID で示される剛体の回転中心の速度ベクトルを返す．3 成分
ということは，**.w()は前 3 成分を返し，.l()は後ろ 3 成分を返す**ということか...

model_.name(bodyID_)　　bodyID_で示される body の剛体名（word 型）を返す
model_.bodyID(reaction)　　reaction（word 型）で示される剛体名の bodyID を返す．
model_.I(bodyID_)　　質量，重心，慣性テンソルを返す
model_.I(bodyID_).m()　　質量を返す．
model_.I(bodyID_).c()　　重心を返す．
model_.I(bodyID_).Ic()　　慣性テンソルを返す．

rigidBody.C

```
os.writeEntry("centreOfMass", c());
```

6. 時間

https://www.openfoam.com/documentation/guides/latest/api/classFoam_1_1Time.html
計算回数 n 回目の時間 t(n)は，計算回数 n 回目の時間刻み dt(n)に対して
t(n)=t(n-1)+dt(n)
となっている（注：t(n)=t(n-1)+dt(n-1)ではない！）ので，時間刻み可変の時には注意が必要．構造計算に渡される時間 t(n)は未来の時間である．

```
#include "createTime.H"

const word &t = runTime.timeName();
const word &t = mesh.time().timeName();

scalar t = runTime.timeOutputValue();
scalar t = mesh.time().timeOutputValue();

scalar deltaT = runTime.deltaTValue();   // 時間刻み
scalar deltaT = mesh.time().deltaTValue();

fileName  path ()
fileName  timePath ()
bool      writeNow ()
```

7. ファイル操作

以下に API guide が示されている．
https://www.openfoam.com/documentation/guides/latest/api/classFoam_1_1fileOperation.html

．ファイルコピー： cp(ファイル名 A，ファイル B，true)，A を B としてコピー

virtual bool cp (const fileName &src, const fileName &dst, const bool followLink=true)

. ファイル名変更： mv(ファイル名 A，ファイル B，true)，A を B として名前変更
virtual bool mv (const fileName &src, const fileName &dst, const bool followLink=false)

. ファイル削除：
virtual bool rm (const fileName &)

. ファイルのパス
virtual fileName filePath (const fileName &)

. ファイルの有無
bool exists (IOobject &io)

8. 型を引数にする記述

8.1 C の場合の swap 関数定義
型に対してそれぞれ関数を用意する.

```
void swapInt(int *i, int *j)
{
        int t;

        t = *j;
        *j = *i;
        *i = t;
}

void swapDouble(double *a, double *b)
{
        double t;

        t = *b;
        *b = *a;
        *a = t;
}
```

8.2 C++の場合の swap 関数の多重定義
2 回書く必要があるが，呼び出しは方に寄らず swap

```
void swap(int *i, int *j)
{
        int t;

        t = *j;
        *j = *i;
        *i = t;
}

void swap(double *a, double *b)
{
        double t;

        t = *b;
        *b = *a;
        *a = t;
}
```

8.3 C++の「Template」による引数を含む関数定義
T クラスの引数はすべて計算する．1 回の記述ですむ．ジェネリック（総称）プログラミング

```
template<class T> void swap(T *a, T *b)
{
        T t;

        t = *b;
        *b = *a;
        *a = t;
}
```

9. const

9.1 const 変数

変数の宣言に const をつけることで、その変数の値が書き換えられないようにできる

```
const int a = 120;
 a = 130;    // コンパイル エラー
```

9.2 const 引数

関数の引数に const を指定すると、その関数の中では値を書き換えることができなくなる

```
class Color
{
public:
      int r, g, b;
};

int Brightness(const Color& a)
{
      return MAX(a.r, MAX(a.g, a.b));
}
```

9.3 const ポインタ

ポインタ変数の宣言に const をつけることで、ポインタの指し示す内容を書き換えることができなくなる

```
Bool IsBlack(const Color * color)
{
     return (color->r == 0 && color->g == 0 && color->b == 0);
}
```

9.4 const メンバ関数

メンバ関数の宣言の末尾に const 修飾子をつけることで、そのメンバ関数を呼び出したときにオブジェクトが変化しないことを宣言できる．このように宣言することで、const オブジェクトに対してもメソッド呼び出しを行うことができる．const オブジェクトに対して呼び出せるメソッドは、const メンバ関数だけ．抜け道はある．mutable

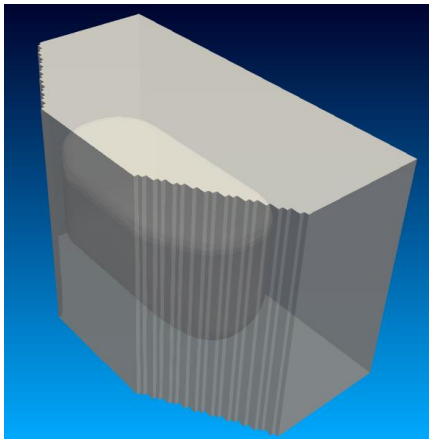
```
#include<iostream>
using namespace std;

class Color
{
private:
      int r, g, b;
public:
      Color() {}                              //この 2 行は constructor の多重定義
      Color(int r0, int g0, int b0)
      {
            r = r0;
            g = g0;
            b = b0;
      }

      ~Color(){ cout << "end" << endl; }// destructor
```

```
    int GetRed()   const { return r; }
    int GetGreen() const { return g; }
    int GetBlue()  const { return b; }

    void SetRed(int n)   { r = n; }
    void SetGreen(int n) { g = n; }
    void SetBlue(int n)  { b = n; }
};

Color func(const Color& color)
{
    Color c;

    c.SetRed(255 - color.GetRed());
    c.SetGreen(255 - color.GetGreen());
    c.SetBlue(255 - color.GetBlue());

        cout << "C1: " << c.GetRed() << ", " << c.GetGreen() << ", " << c.GetBlue() << endl;

    return c;
}

int     main()
{
        Color     a(255,1,1);

        cout << "M1: " << a.GetRed() << ", " << a.GetGreen() << ", " << a.GetBlue() << endl;
        a = func(a);
        cout << "M2: " << a.GetRed() << ", " << a.GetGreen() << ", " << a.GetBlue() << endl;

        return 0;
}
```

この話の意味合いは，オープンソースにおいて，他人がこの関数を呼び出した時に，中身の変数を変更されないようにするためであるが，これによって，いちいちコンパイルエラーが起こったりする．

10. メッシング

．snappyHexMesh の境界線の平滑化（snapControls）は，壁境界（type wall）にしかできない（もしくは，重合格子境界（type overset）にはできない）．境界タイプを指定しないままの（blockMesh で設定せず，snappyHexMesh で作成する）場合，自動的に壁境界になる．



外側は，type overset なので平滑化できず，内側のボディは type wall なので平滑化できる．仕様が残念...

．snappyHexMesh の addLaylerControls は，滑らかな（鈍角な？）面にしか境界層を作ってくれない．概ね，円筒のような形状以外はうまく境界層はできない．四角ですら難儀する．

．境界層の設定はかなり難しい．
addLayersControls の
　　featureAngle 360;
設定で鋭角（例えば蝶の翼端）の部分にもつぶれない境界層が作れるようになる．また，blockMesh 時のそもそものメッシュも細かくないと切れない．

．snappy と気液二相流の p_rgh solver の相性が強烈に悪い件

気液二相流で，snappy はなぜか使えない．tutorial の boatAndPropeller でも，わざわざ refineMesh で細分化している．snappy を使った重合格子を，snappy を使っていない外側の計算空間とマージするのはいいようだ．

(1) snappy を使うと，それまで使えていた DILU が定義されていないとのエラーがでる．恐らく，オーバーロードされていて，そっちで定義されていない．

```
p_rgh
{
    solver          PBiCGStab;
    preconditioner  DILU;
    tolerance       1e-9;
    relTol          0.01;
}
```

(2) preconditioner を，DIC などにすると，かろうじて動くが一瞬で発散．
(3) solver を PCG に変えると，数回計算できるがすぐ発散

11. 流体力学計算関係

9.1 混相流計算（overInterDyMFoam, ）
クーラン係数が，全体と，二層流境界面で二つ定義されている．これに基づいて時間刻みを決定できる．

```
maxCo           0.5;    /*  全体のクーラン数上限  */
maxAlphaCo      0.5;    /* 水面（0<alpha.water<1）のクーラン数上限*/
```

12．　境界条件設定

．乱流条件

乱流モデル（PENGUIN）
http://penguinitis.g1.xrea.com/study/note/turbulence_model.pdf

Turbulence free-stream boundary conditions:
https://www.cfd-online.com/Wiki/Turbulence_free-stream_boundary_conditions

Cradle 伊丹 隆夫
https://www.cradle.co.jp/media/column/a343

．混相流流速

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (10 0 0);

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    inletSide
    {
        type            fixedValue;
        value           $internalField;
    }

    outletSide
    {
        type            outletPhaseMeanVelocity;
        alpha           alpha.water;
        Umean             10;           // 流速の平均値だが，tutorial と符号が違う．
```

```
        value           $internalField;
    }

    atmosphere
    {
        type            pressureInletOutletVelocity;
        tangentialVelocity $internalField;
        value           uniform (0 0 0);
    }

    flyingfish
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
}
```

．非定常設定

現時刻 t に依存した入口速度設定．実行時のコンパイル(Run-Time Code Compilation)により設定される．

```
    inletSide
    {
        type            codedFixedValue;
        value           uniform (0 0 0);
        redirectType    sinVelocity;

        code
        #{
            fvPatchField<vector> f
            (
                patch().lookupPatchField<volVectorField, vector>("U")
            );

            const scalar t = this->db().time().timeOutputValue();      // 現在の時間の取得
            const scalar pi = constant::mathematical::pi;              // π
            scalar ux;

            if( t < 1.0 )       ux = 0.5 * ( 1.0 - cos(pi*t) ) / 2.0;
            else                ux = 0.5;

            forAll(patch().Cf(), i)                 //パッチループ：流入口パッチに ux を設定
            {
                f[i] = vector(ux, 0., 0.);
            }
            operator==(f);
        #};
    }
```

なお．

forAll(p, i)

は，

for(int i=0; i< p.size(); i++)

という意味．

．圧力の単位がおかしい話

openFOAM の単相流非圧縮性ソルバでは，圧力は単位密度当たりの圧力（小文字の p）になっている．つまり，

P/ρ　= [Pa] / [kg/m^3] = m^2 /s^2

よって，圧力の境界条件ファイルには以下のように書かれている．

```
/*--------------------------------*- C++ -*----------------------------------*¥
| =========                 |                                                 |
| ¥¥          /   F ield            | OpenFOAM: The Open Source CFD Toolbox           |
|  ¥¥        /    O peration        | Version:    v2006                               |
|   ¥¥      /     A nd              | Website:    www.openfoam.com                    |
|    ¥¥/       M anipulation  |                                                 |
¥*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    "(stationarySides|ground)"
    {
        type            zeroGradient;
    }
.....
```

内部の計算は，constant/transportProperties で動粘性（nu）のみが設定されており，密度は定義されていない．圧力（Pa）の表示のためには，Paraview で密度をかけて表示する必要がある．
連成計算の時には，constant/dynamicMeshDict で
rho rhoInf;
rhoInf 1.293;
として設定しているようだ．後日要確認．

なお，NS 方程式としては，

$$\begin{bmatrix} \frac{\partial(\rho u)}{\partial t}+\frac{\partial(\rho uu)}{\partial x}+\frac{\partial(\rho uv)}{\partial y}+\frac{\partial(\rho uw)}{\partial z} \\ \frac{\partial(\rho v)}{\partial t}+\frac{\partial(\rho vu)}{\partial x}+\frac{\partial(\rho vv)}{\partial y}+\frac{\partial(\rho vw)}{\partial z} \\ \frac{\partial(\rho w)}{\partial t}+\frac{\partial(\rho wu)}{\partial x}+\frac{\partial(\rho wv)}{\partial y}+\frac{\partial(\rho ww)}{\partial z} \end{bmatrix} = \begin{bmatrix} \rho f_x-\frac{\partial P}{\partial x}+\mu(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2})+\frac{1}{3}\mu\frac{\partial}{\partial x}(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}) \\ \rho f_y-\frac{\partial P}{\partial y}+\mu(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2})+\frac{1}{3}\mu\frac{\partial}{\partial y}(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}) \\ \rho f_z-\frac{\partial P}{\partial z}+\mu(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2})+\frac{1}{3}\mu\frac{\partial}{\partial z}(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}) \end{bmatrix}$$

ではなく，

$$\begin{bmatrix} \frac{\partial u}{\partial t}+\frac{\partial(uu)}{\partial x}+\frac{\partial(uv)}{\partial y}+\frac{\partial(uw)}{\partial z} \\ \frac{\partial v}{\partial t}+\frac{\partial(vu)}{\partial x}+\frac{\partial(vv)}{\partial y}+\frac{\partial(vw)}{\partial z} \\ \frac{\partial w}{\partial t}+\frac{\partial(wu)}{\partial x}+\frac{\partial(wv)}{\partial y}+\frac{\partial(ww)}{\partial z} \end{bmatrix} = \begin{bmatrix} f_x-\frac{\partial p}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)+\frac{\nu}{3}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right) \\ f_y-\frac{\partial p}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right)+\frac{\nu}{3}\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right) \\ f_z-\frac{\partial p}{\partial z}+\nu(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2})+\frac{\nu}{3}\frac{\partial}{\partial z}(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}) \end{bmatrix}$$

where
$\nu=\frac{\mu}{\rho},\ \mathrm{p}=\frac{P}{\rho}$

として計算している．こうすると，支配方程式の計算において係数は動粘性係数のみになる．

13. 重合格子計算

13.1 重合格子計算アルゴリズム

とりあえず，以下を正しい解説として，参照して説明する（間違った力学を解説しているものも web 上にたくさんあるので，最終的にはプログラムを見るしかない）．

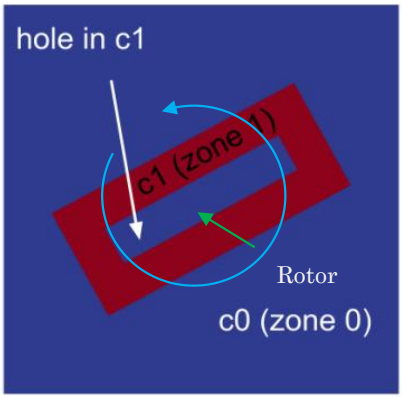
Description of the overset mesh approach in ESI version of OpenFOAM

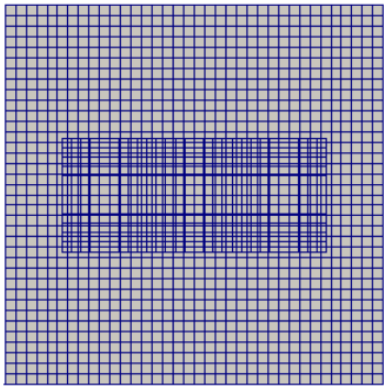
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2019/Tisovska_Petra/report_TisovskaPetra.pdf

チュートリアルのシンプルロータを例にすると，

tutorial/incompressible/overPimpleDyMFoam/simpleRotor

下図左のように，青が外側の計算空間（background: zone c0），赤が重合格子の計算空間（overset: zone c1）であり，重合格子の空間には，ロータの穴が開いており，重合格子全体が回転する．格子は，ロータの角度が 0 のとき，下図右のようになっている．





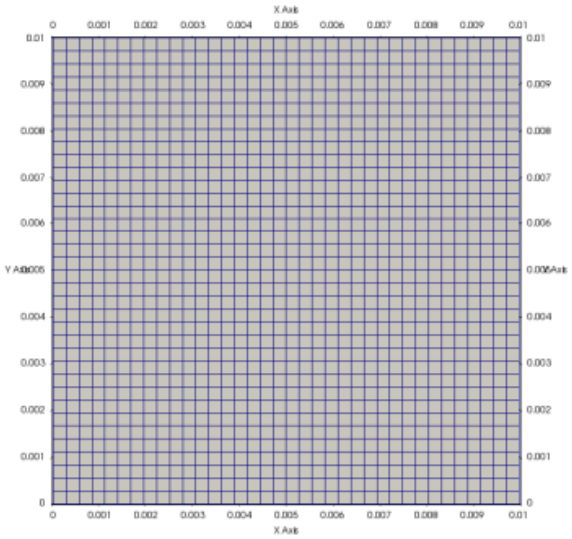
重なっている領域の物理量（流速と圧力）を収束計算により同じになるように求めるわけであるが，openFoam では以下の手法を採用している．まず，格子の領域は，以下の三つに分けられる．

計算領域（Calculated）：運動方程式を解く領域．ただし，実際は，重なっている領域と重なっていない領域に，さらに二つに分けられるはずであるが，どの変数で認識しているかは不明．zoneID でどの領域かは分かる．

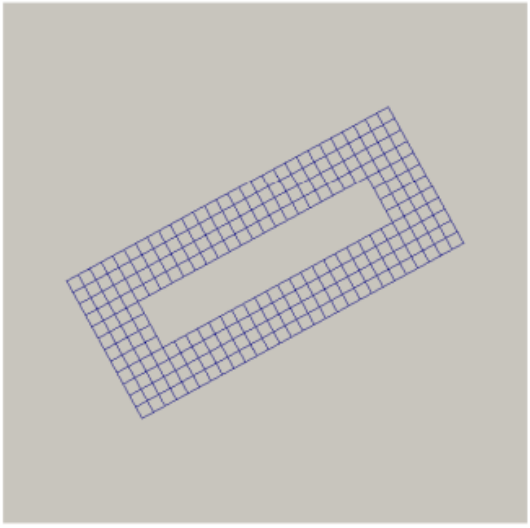
内挿領域（Interpolated）：他方の領域（overset なら background，background なら overset）の要素の近傍から内挿によって求められる領域．重合格子領域が複数ある時にどうなるかは不明．

空洞領域（Holes）：ここでは計算しない領域．おそらく，以前の値がそのまま残っている．

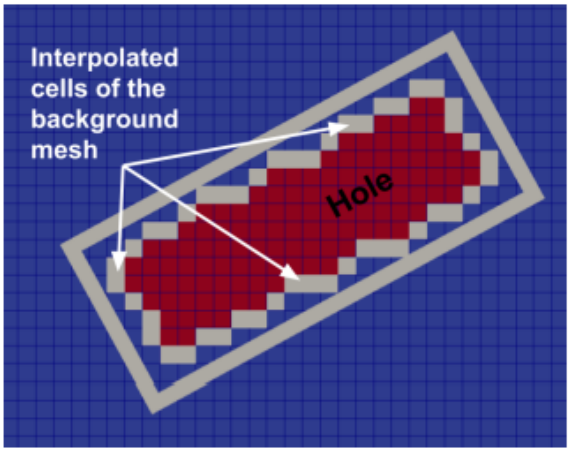
ということは，収束計算するのは，重なっている計算領域か．．．？　内挿部分を境界条件にして，重合部分が一致するまで計算？



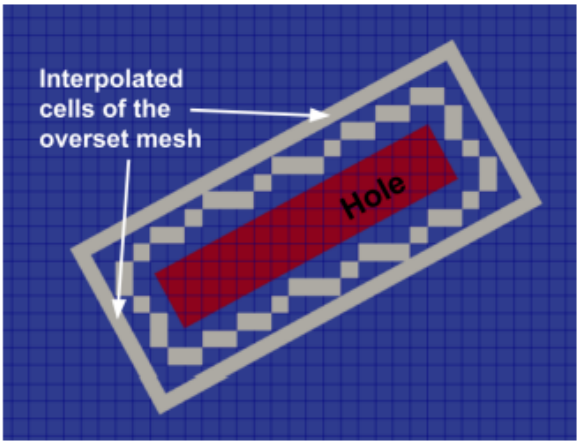
(a) Background mesh



(b) Overset mesh



(a) Hole (red cells) and interpolated cells (white cells) defined for background mesh



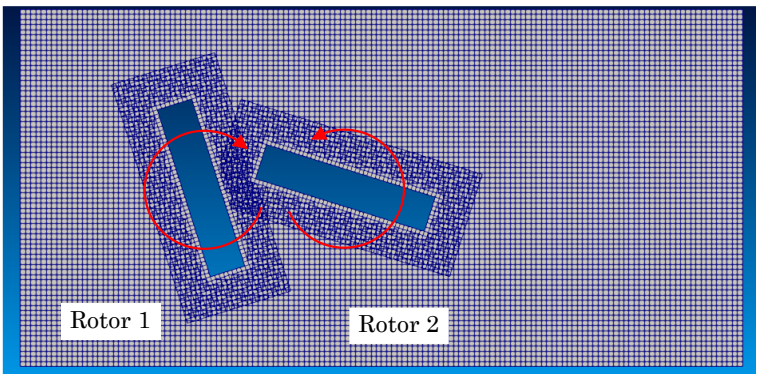
(b) Hole (red cells) and interpolated cells (white cells) defined for overset mesh

これらの領域は，
system/fvSchemes
の
oversetInterpolation
{
　　method　　cellVolumeWeight;
}
によって，割り当てられている．

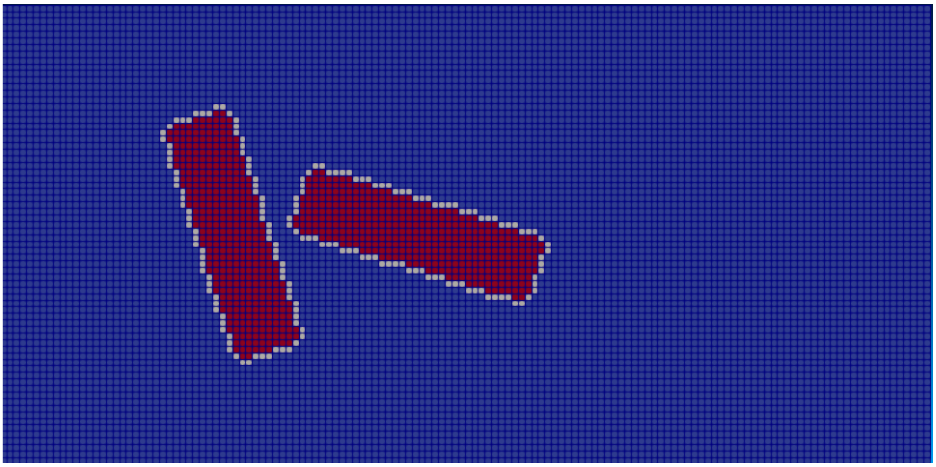
13.2 複数の重合格子がある場合

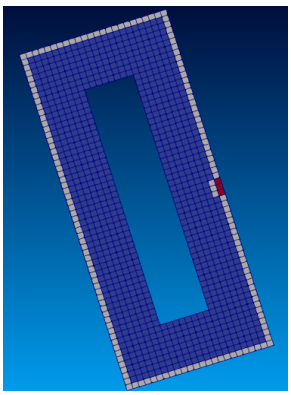
チュートリアルの two シンプルローターズで考える．
tutorial/incompressible/overPimpleDyMFoam/simpleRotor

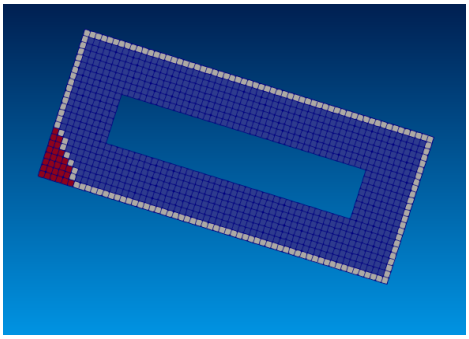
上記のシンプルロータが横に二つ並んでいて，ロータどおしは重ならない（衝突しない）が，二つの重合格子領域どうしが重なる瞬間はある．その瞬間を下図に示す．



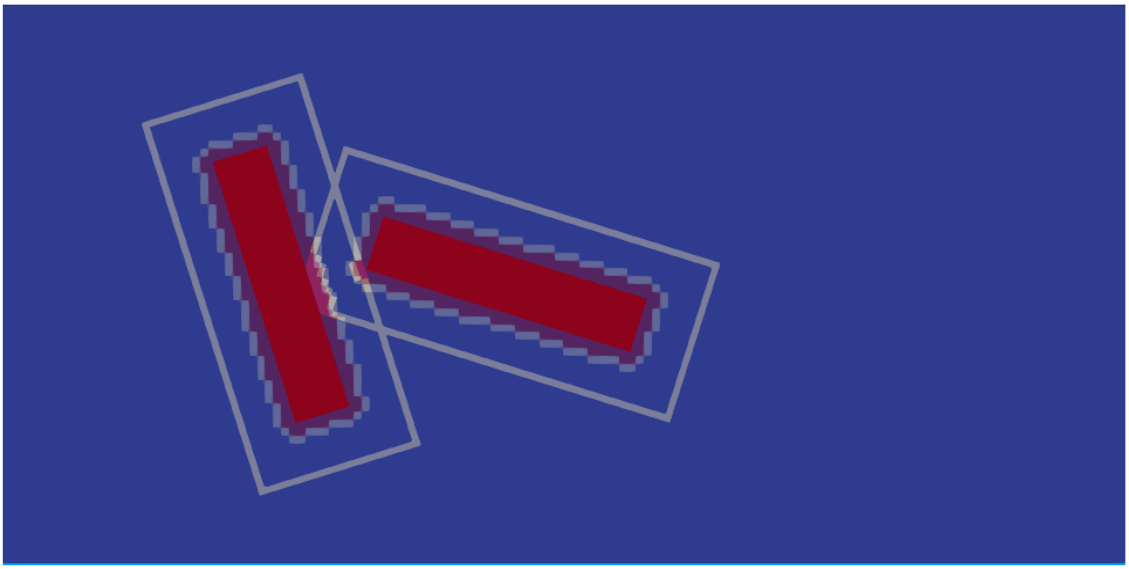
この瞬間の領域は，Filters の Threshold を使って zoneID を分け，cellTypes（青： 0 =計算領域，白：1=内挿領域，赤：2=空洞領域）で表示すると以下のようになっている．







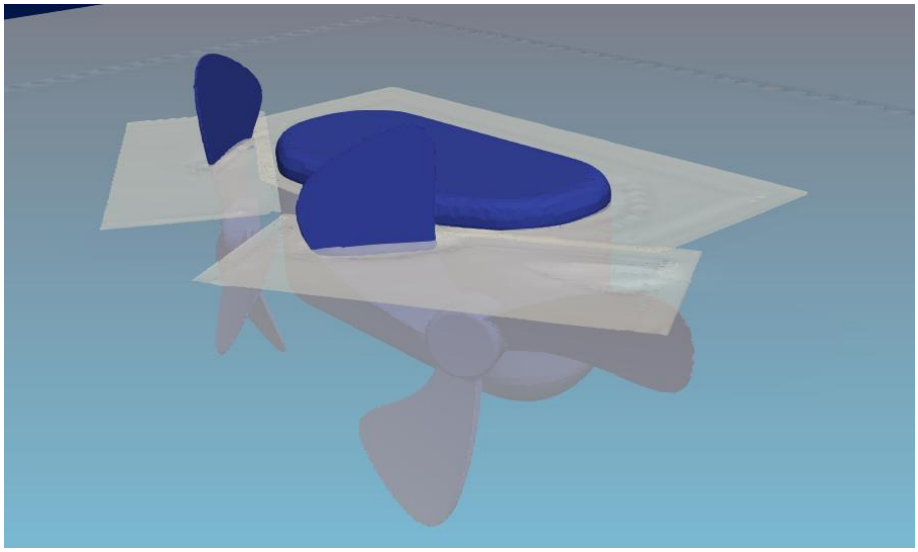
重ねて，透過してみるとこんな感じ．

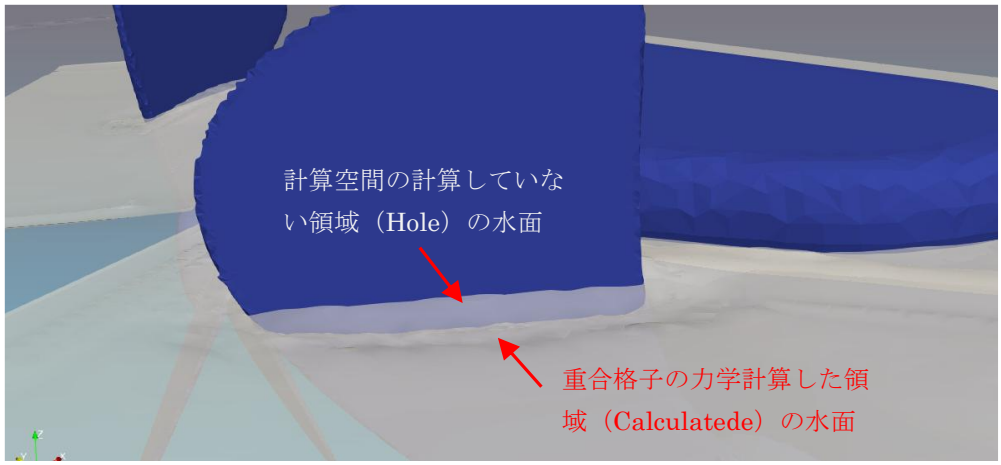


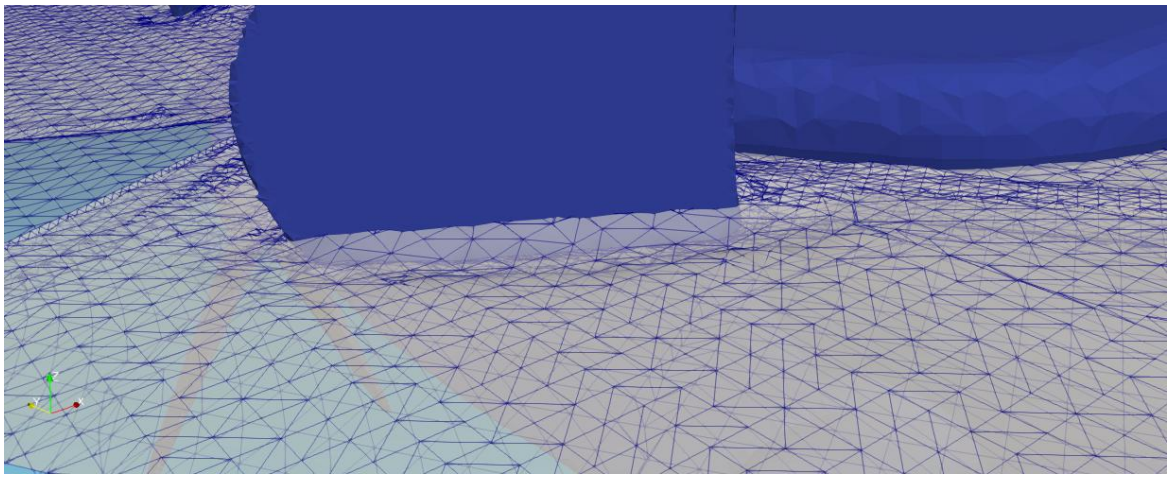
もしかすると，内挿する領域で囲まれていなければないのかも...
内挿領域同士が重なるとどうなる？

13.3 風呂ロボの praview での表示の話

風呂ロボの計算では，40ms で以下のような計算結果が得られる．水面の色の濃くなっているところが，重合格子領域の水面まで二重に表示している場所である．結局，paraview が計算していない hole の領域の結果も表示してしまうとこに問題がある...



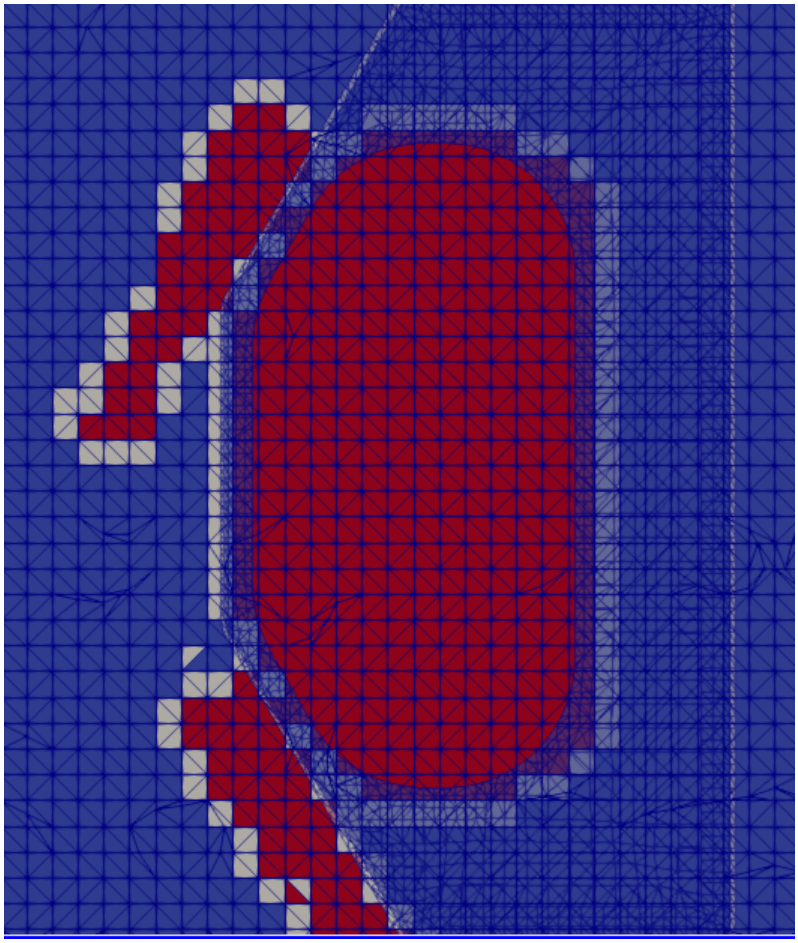




上記同様，cellTypes の 2（空洞領域）を，filters の Threshold でフィルタリング（マスク処理）することで解消できるが，透過処理すると，二重に表示されて濃くなる．

13.4 風呂ロボの重合格子の話

以下のような重なり方は恐らくダメ．ボディの内挿が，(a)外側の内挿に重なり続けるとか，(b)外側の空洞領域かつペラの空洞領域で内挿できないとか．



13.5 どれとどれを重合させるか.

全ての重合格子をどうしを計算している.

例えば，log をみると，

Creating mesh-to-mesh addressing for region0 and region0 regions using cellVolumeWeight

    Overlap volume: 0.023727022

Creating mesh-to-mesh addressing for region0 and region0 regions using cellVolumeWeight

    Overlap volume: 0.0011724447

Creating mesh-to-mesh addressing for region0 and region0 regions using cellVolumeWeight

    Overlap volume: 0.0011722652

.....

cellVolumeWeight : detected 26 mesh regions after overset

のように示されている．n 個の格子がある場合，${}_nC_2$ だけ，creating mesh をすることになる．重合格子同士が重なっていない場合には，overlap volume が 0 になる.

14. デバック関係

．メッシュが歪んで計算が止まったと思われるとき

> checkMesh

で品質をチェックできる.

品質に問題がないときの出力：

Time = 0.004

Checking geometry...

    Overall domain bounding box (-0.5 -0.5 -0.5) (0.5 0.5 0.5)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (-5.44342e-17 5.91386e-17 -1.20854e-16) OK.

Max cell openness = 3.17799e-16 OK.
Max aspect ratio = 3.08549 OK.
Minimum face area = 9.61994e-07. Maximum face area = 0.00015625. Face area magnitudes OK.
Min volume = 1.63385e-09. Max volume = 1.95313e-06. Total volume = 1.06391. Cell volumes OK.
Mesh non-orthogonality Max: 31.0844 average: 2.08
Non-orthogonality check OK.
Face pyramids OK.
Max skewness = 0.685528 OK.
Coupled point location match (average 0) OK.

品質に問題があるときの出力：

Time = 0.168

Checking geometry...
Overall domain bounding box (-0.5 -0.5 -0.5) (0.5 0.5 0.5)
Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
Mesh has 3 solution (non-empty) directions (1 1 1)
Boundary openness (-1.86722e-18 5.99605e-17 -8.4141e-17) OK.
Max cell openness = 2.91064e-16 OK.
Max aspect ratio = 3.89574 OK.
Minimum face area = 5.28047e-08. Maximum face area = 0.00015625. Face area magnitudes OK.
Min volume = 4.95242e-10. Max volume = 1.95313e-06. Total volume = 1.06391. Cell volumes OK.
Mesh non-orthogonality Max: 44.2709 average: 3.54746
Non-orthogonality check OK.
***Error in face pyramids: 28 faces are incorrectly oriented.
<<Writing 28 faces with incorrect orientation to set wrongOrientedFaces
Max skewness = 2.52803 OK.
Coupled point location match (average 0) OK.

Failed 1 mesh checks.

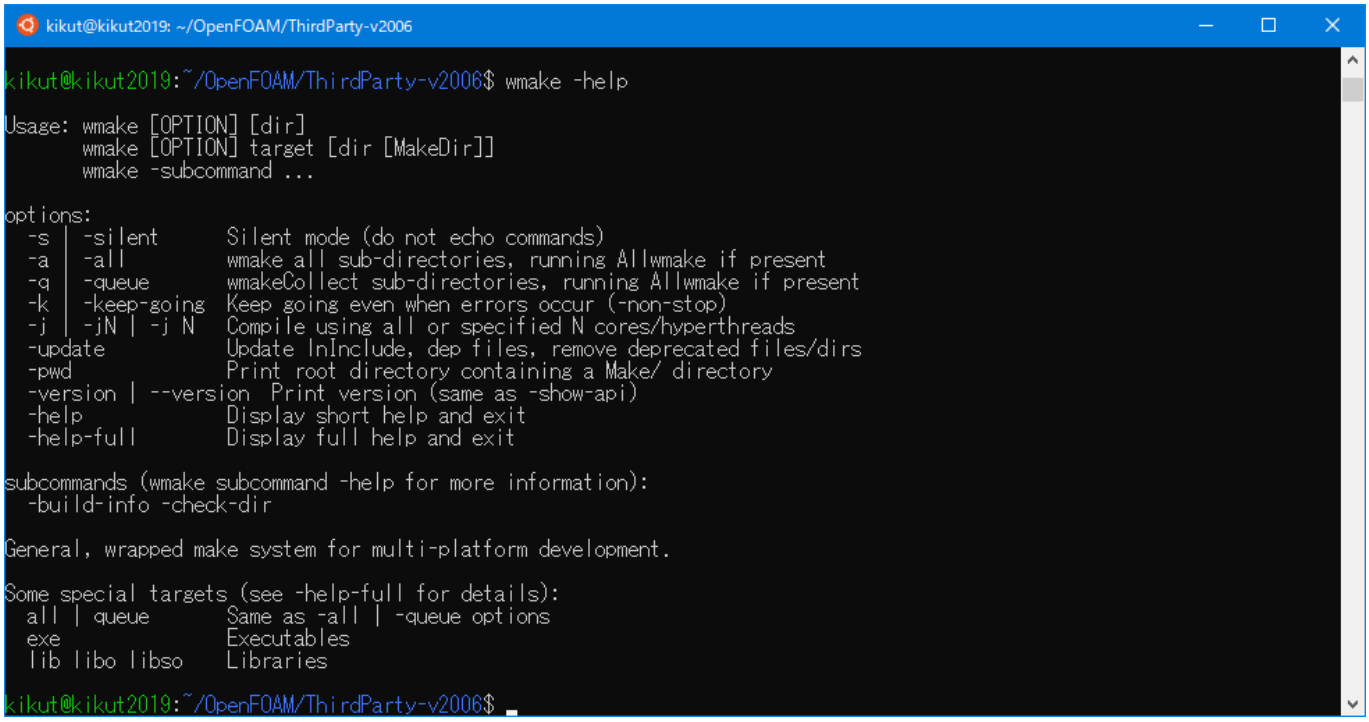
. model_.X0(bodyID_)の姿勢と回転中心は，異なるタイミングで更新されている．例えば，restraint のバネ設定などで，Constructor の時点では，姿勢は初期化されているが，回転中心は初期化されておらず，0 になっている．ある意味，どちらも I, 0 に初期化されているのかもしれない．

15. Linux コマンドあれこれ

バックグラウンドでプロセスを実行
> nohup コマンド > ファイル名 &

圧縮
> tar -zcvf xxxx.tar.gz directory
解凍
> tar -zxvf xxxx.tar.gz

openfoam のファイルのコンパイル
> wmake



ライブラリの作成は，

> wmake libso

場所は，以下．ここからライブラリが呼び出される．openfoam 側のライブラリのパスより，USER のライブラリのパスの方が先に記述されているので，同じファイルがあった場合，USER 側が優先される．

$FOAM_USER_LIBBIN/

paraview install

> sudo apt install paraview

16. 定義ディレクトリあれこれ

ユーザー名が kikut，Ubuntu8.4，OpenFoam v2012 の例である．

ユーザー作成ライブラリの保存場所

$FOAM_USER_LIBBIN

/home/kikut/OpenFOAM/kikut-v2012/platforms/linux64Gcc63DPInt32Opt/lib

ユーザー作成ソルバの保存場所

$FOAM_USER_APPBIN

/home/kikut/OpenFOAM/kikut-v2012/platforms/linux64Gcc63DPInt32Opt/bin

ライブラリソースファイル保存場所

$FOAM_SRC

/opt/OpenFOAM/OpenFOAM-v2012/src

ソルバソースファイル保存場所

$FOAM_SOLVERS

/opt/OpenFOAM/OpenFOAM-v2012/applications/solvers

17． バク情報あれこれ．ご注意

. rad と明記されていて，実際は deg は多々あり．

. 回転変換は，グローバル座標系になる場合が多々ある．使い方が悪いのかバクなのかはわからない．

. 絶対値は，mag(x) を使う．クラスにもよるかもしれないが，absf は使えず，abs は正しくない答えを返してくる．ex. abs(-0.1)=0)

. CAD ファイルのスケールは，1/1000 になっている．時と場合により 1/100 の報告もあり．ただし，以下でスケール変換できる．

- surfaceConvert -scale 0.1 A.stl B.stl

A.stl を 1/10 にして B.stl に．

. rigidBodyDynamics/rigidBodyModelState で以下（v2006）が定義されている．

```
//- Return access to the joint position and orientation
inline scalarField& q();

//- Return access to the joint velocity
inline scalarField& qDot();

//- Return access to the joint acceleration
inline scalarField& qDdot();
```

q()は 7 成分のベクトルで，最初の 3 成分は重心位置ベクトルになっていて，残り 4 成分が orientation とのことであるが，後者の定義が分からない．ノルムが 1 でないので，quaternion ではない．ロドリゲスもどきでもない．たぶん，間違い．何かで正規化すると quaternion になるのかもしれない．単純なノルム 1 への正規化ではなかった．4 元数の姿勢表現を理解していないプログラムのように思われる．

v4.1 の説明文には，qDot()は quaternion である旨記載されているが，速度に記述するのは間違い．これ以降ではこの記述は消されている．

rigidBodyDynamics¥rigidBodySolvers¥rigidBodySolvernewRigidBodySolver.C

には，以下の記載が残されている．quaternion の変数は消したということ？

```
//- Return the motion state
inline rigidBodyModelState& state();

//- Return the current joint position and orientation
inline scalarField& q();

//- Return the current joint quaternion

//- Return the current joint velocity
inline scalarField& qDot();

//- Return the current joint acceleration
inline scalarField& qDdot();
```

. 縦，横ベクトルの定義も極めてあいまい．

姿勢行列は，一般的なものと違い，転置されており，

$$R = \begin{bmatrix} \ldots x \ldots \\ \ldots y \ldots \\ \ldots z \ldots \end{bmatrix}$$

のように，x, y, z 軸が並べられている．これより，物体座標系のベクトル $\boldsymbol{a} = (a_x, a_y, a_z)$ は，横ベクトルとなり，基準座標系で表すと，

$$\boldsymbol{a}R = (a_x, a_y, a_z) \begin{bmatrix} \ldots x \ldots \\ \ldots y \ldots \\ \ldots z \ldots \end{bmatrix}$$

となる．一般的な記述は，右から縦ベクトルをかけるので，無理やり一般的な記述に直すと，

$$R^{-1}\boldsymbol{a}^T = \begin{bmatrix} x & y & z \\ & & \end{bmatrix}\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$$

となる．また，上記計算は

vector a;

として宣言すると，”&”（内積，ベクトル演算）でどちらもできる．

本来，左の列の数と，右の行の数が一致しないと計算できないはずであるが，あいまいになっている．

$$\begin{pmatrix} a_x & a_y & a_z \end{pmatrix}\begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x b_x + a_y b_x + a_z b_x$$

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}\begin{pmatrix} b_x & b_y & b_z \end{pmatrix} = \begin{bmatrix} a_x b_x & a_x b_y & a_x b_z \\ a_y b_x & a_y b_y & a_y b_z \\ a_z b_x & a_z b_y & a_z b_z \end{bmatrix}$$

という概念がない．縦か，横か分からないベクトルを右からでも左からでもかけられる．

実際の計算結果は，以下の通り．

program:

```
Tensor<scalar> A;

A.xx() = cos( 3.141592/4 );     A.xy() =-sin( 3.141592/4 );     A.xz() = 0.0;
A.yx() = sin( 3.141592/4 );     A.yy() = cos( 3.141592/4 );     A.yz() = 0.0;
A.zx() = 0.0;                   A.zy() = 0.0;                   A.zz() = 1.0;

vector a={1,0,0};

Info << "A = " << endl << A << endl;
Info << "a = " << endl << a << endl;
Info << "Aa = " << endl << (A&a) << endl;
Info << "aA = " << endl << (a&A) << endl;
```

result:

```
A =
(0.707107 -0.707107 0 0.707107 0.707107 0 0 0 1)
a =
(1 0 0)
Aa =
(0.707107 0.707107 0)
aA =
(0.707107 -0.707107 0)
```

．ニューマークベータの係数に関して

Newmark.C の Constructors に

```
Foam::RBD::rigidBodySolvers::Newmark::Newmark
(
    rigidBodyMotion& body,
    const dictionary& dict
)
:
    rigidBodySolver(body),
```

```
    gamma_(dict.getOrDefault<scalar>("gamma", 0.5)),
    beta_
    (
        max
        (
            0.25*sqr(gamma_ + 0.5),
            dict.getOrDefault<scalar>("beta", 0.25)
        )
    )
{}
```

と記載されているが,

$$\beta = \frac{(\gamma + 0.5)^2}{4}$$

が無条件安定の式では？　分岐の意味が分からない．条件によっては設定した beta の値ではなくなっていることに注意する必要がある．

．陰解法の間違いに関して

Newmark しか検証していないが，恐らくすべての陰解法が間違っている．流れとしては，陽解法のように解かれている．
例えば，NewmarkBeta 法は，n+1 の位置と速度は，γ=0.5，β=0.25 のとき,

$$v_{n+1} = v_n + \frac{a_n + a_{n+1}}{2} dt$$

$$d_{n+1} = d_n + v_n dt + \frac{1}{2}\frac{a_n + a_{n+1}}{2} dt^2$$

であり，加速度の変化が dt の間で線形であると仮定している．未来の a_{n+1} は，例えば，運動方程式

$$Ma_{n+1} + C(v_n + \frac{a_n + a_{n+1}}{2} dt) + K(d_n + v_n dt + \frac{1}{2}\frac{a_n + a_{n+1}}{2} dt^2) = f_{n+1}$$

から求めるが，openFoam では,

$$v_{n+1} = v_n + \frac{a_{n-1} + a_n}{2} dt$$

$$d_{n+1} = d_n + v_n dt + \frac{1}{2}\frac{a_{n-1} + a_n}{2} dt^2$$

として，陽的に求めている．
連成解析の場合には，上記運動方程式の f_{n+1} を最小二乗法などで推測し，a_{n+1} を収束計算で求めることになるが，非線形性が強いとうまくいかない．菊池研作成ライブラリの NewmarkCOMimplicit がこれを行っている．本来は，流体力も含めて収束計算を行う必要があるが，構造計算コストに対して，流体計算コストは指数的に大きいので検討が必要である．ANSYS などはこれを行っているが，時間がかかりすぎるのと，液体では収束もし難い．

．剛体の力学における加速度，躍度の緩和に関して

rigidBodyMeshMotion ライブラリなどで用いられている以下の緩和係数であるが,
accelerationRelaxation 0.9;
accelerationDamping　0.9;
力学的には完全な間違いなので使わない方がよい．マニュアルらしき HP によると，以下のように解釈できる：
accelerationRelaxation=0.9 は，加速度を 90%に緩和し，数値発散を防ぐことを目的としているが，10m/s^2 の加速度が 9m/s^2 になることを意味しており，自由落下すら正しく計算できない．
accelerationDamping=0.9 は，躍度が同様に 90%になることを意味している．ダンパという名前がついているが，1 が緩和しないことであり，0 が等加速度にすることであるので，一般的なダンパとは意味が逆である．速度制御をする prescribedRotation などにおいては，剛体運動による流体反力とが振動して数値発散するのを防ごうとしている．これらは成り立つのは，定常になり等速運動（加速度＝躍度＝０）になった流れだけである．
ただし，プログラムを見ると,

```
void Foam::RBD::rigidBodyMotion::forwardDynamics
(
    rigidBodyModelState& state,
```

```
    const scalarField& tau,
    const Field<spatialVector>& fx
) const
{
    scalarField qDdotPrev = state.qDdot();
    rigidBodyModel::forwardDynamics(state, tau, fx);
    state.qDdot() = aDamp_*(aRelax_*state.qDdot() + (1 - aRelax_)*qDdotPrev);
}
```

aRelax_→ accelerationRelaxation
aDamp_→ accelerationDamping
であるので，マニュアルによる説明も間違っている．
前回との加重平均加速度に対して緩和係数をかけている計算のようだ．
しかもこれだと時間刻み可変に対応できないのでは？

．作用反作用の概念がほとんどない．よって，マルチボディで複数の剛体を組み合わせて計算する場合，反トルクや，反モーメントを設定するのが，極めて面倒になる．多くの場合，ライブラリを書き換えることになる．

．rigidBodyModel.H に以下の記載があるので，これが true の時だけ quaternion を計算するということか？

```
    //- Return true if any of the joints using quaternions
    inline bool unitQuaternions() const;
```

．関数によって，順番が逆になっているので注意．
rigidBodyDynamics においては，
fx[i][6] = （モーメント，力）
model_.v(bodyID_)= （角速度，速度）
state.q() = （位置，姿勢（4 元数））

．重力係数関係のトラップ
ソルバによって，重力の設定が異なるようだ．おそらく，
．constant/g ファイルによる重力設定は，多相流では必要で，単相流（重力がつりあっている流れ）では必要ない．剛体計算をする場合には，単相流の場合には，dynamicMeshDict で指定する．多相流の場合には，dynamicMeshDict に書いても，g ファイルの内容で上書きされる．つまり，単相流（pimple 等）系の重力の設定は，dynamicMeshDict ファイルに
g (0 0 -9.8)
の指定を書く必要がある．
． controlDict の function 計算などでは，単相流では重力指定が必要で，多相流では必要ない．

．密度のトラップ
非圧縮計算の場合には，圧力は P/ρ として計算されている場合があるため，単位が異なっている．この場合には，力を別途計算する場合には，密度を別に指定する必要がある．pimple における function で力を計算する場合など．
剛体計算をする場合には，単相流系の場合には，dynamicMeshDict ファイルに，例えば，
rho rhoInf;
rhoInf 1.293;
を記載する必要がある．

．表面張力のトラップ
transportProperties のファイルに，表面張力（水と空気間）の値として，
sigma 0.007;
と書かれてい部場合と，
sigma 0.07;

と書かれている場合があるが，正解は後者．

．スペルミスは多々あり．
initialise -> initialize
rigidBodyMeshMotion クラスの定義が，rigidBodyMotion になっている．dynamicMeshDict ファイルで呼び出されているソルバ，

motionSolver　　rigidBodyMotion;

は，実は，rigidBodyMeshMotion である．

．剛体の運動方程式（rigidBodyDynamics 系クラス）のトラップ
物体座標系の原点（RBD では，これを回転中心 CoR と呼んでいる）で運動方程式が立てられており，剛体の自由運動に対応していないと思われる．このため，菊池研ライブラリでは COM 系クラスを作ってある．ex. NewmarkCOMimplicit

．インストール関係のトラップ
これも地味に引っかかるので，PATH などの設定は要注意
インストール先のディレクトリ名がちょいちょい変わっているので，別のコンピュータのエイリアスをコピーすると痛い目に会う．ex.:
¥opt¥OpenFOAM¥OpenFOAM-v2006
¥opt¥openfoam¥OpenFOAM-v2006
どうやら，v2006 はこの 2 パターンがあるようだ．そして，v2106 からは，以下に変更された．
¥usr¥lib¥openfoam¥openfoam2106

．クーラン数による動的時間刻みのトラップ
system/controlDict の
adjustTimeStep　true;
でクーラン数に依存した時間刻みを調整できるが，剛体計算をした時に，正しく計算できているか疑問が残る．
model_.v(bodyID_,p)などの速度計算が正しく行われていないか，ニューマークベータの積分に失敗してるか，この辺に原因があるように思われる．また，ファイルの保存タイミングにもこの時間刻みが影響されるので，注意が必要である．
剛体計算を伴う連成解析の場合には，
adjustTimeStep　false;
として時間刻みを一定にしておくのが，無難である．クーラン数が 1 を超えないように常に監視しておく必要はある．
→この問題は，COM 系で一応解決しているが，openFoam では，ある程度誤差を許容しているようだ．例えば，1ms 毎データを保存するとして，
0.001
というディレクトリに保存されるわけであるが，
uniform/time
をみると，実際の時間は，
value 0.0010000000000000197
となっている．これは時間刻みの変更や，経過時間でそこそこ大きくなってくる．解析などでは問題はほとんど起こらないが，プログラミングなどでは注意が必要である．”t=1 ms のとき“，を意味する：
if(mag(t – 0.001) < 1.0e-32) ***
などは失敗する．

．WSL のファイル名の大文字小文字のトラップ

Windows の OS アーキテクチャは，ファイル名の大文字と小文字を認識しないため，OpenFOAM のプログラムの実行には注意が必要である．例えば，**Windows 上のメモリでは**，同じディレクトリ内に

aaa.H

Aaa.H
という二つのファイルを作れない．また，

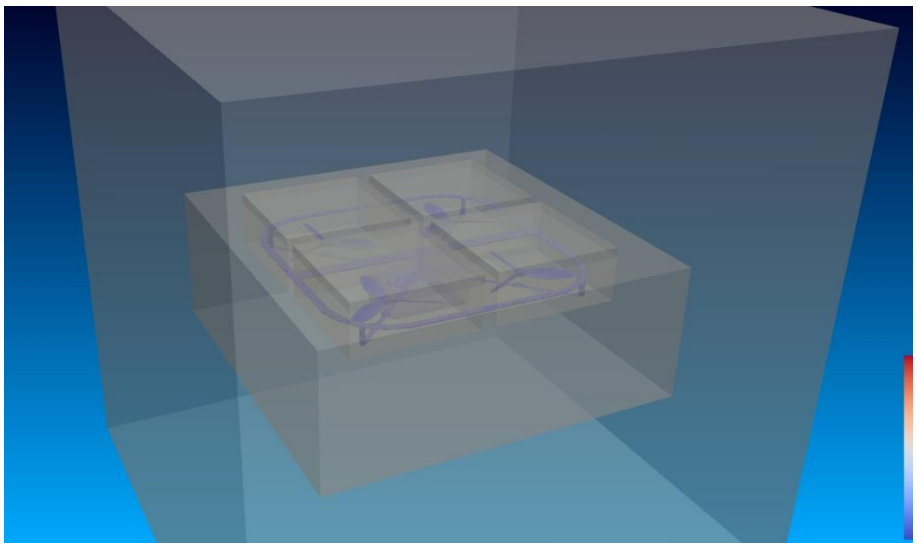
pimpleFoam/include/aaa.H
simpleFoam/include/Aaa.H
というファイルが存在し，pimpleFoam/include/内のプログラムで，Aaa.H を呼び出しても，aaa.H が読み込まれる（フルパス指定すればこれは回避できる）．さらに，

ALlrun
とタイプミスしても，Allrun を実行することがでる．
WSL では，**Linux 側のメモリ**を以下の領域に割り当てている．

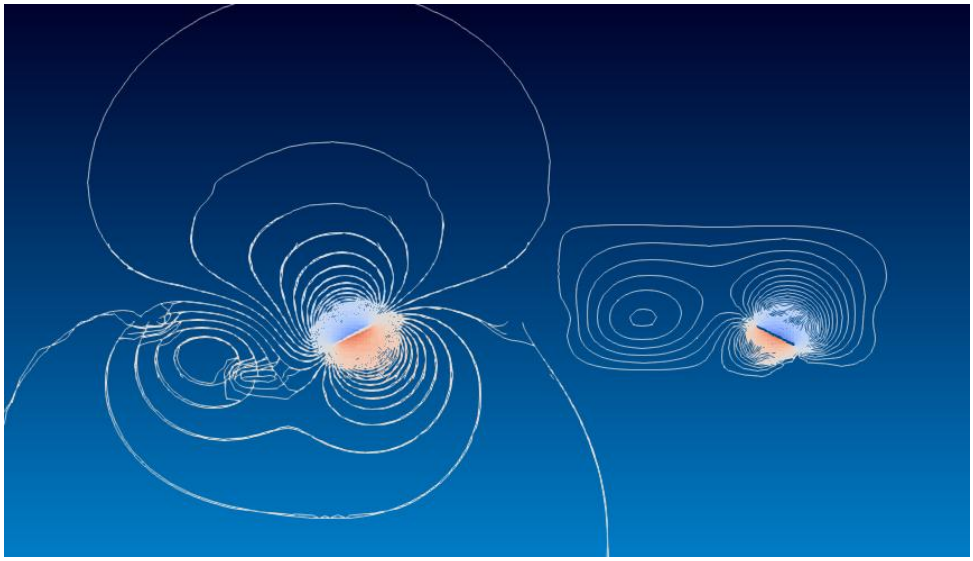
¥¥wsl$¥Ubuntu-18.04¥

これ以下の領域は，ファイルの大文字と小文字を認識でき，上記のトラブルを回避できる．よって，ソルバやライブラリのメイクは，必ずこの領域で行う必要がある．windows のメモリでメイクを行うと，コンパイルできないか，できてもインクルードファイルなど異なるファイルを読み込んでいる可能性がある．

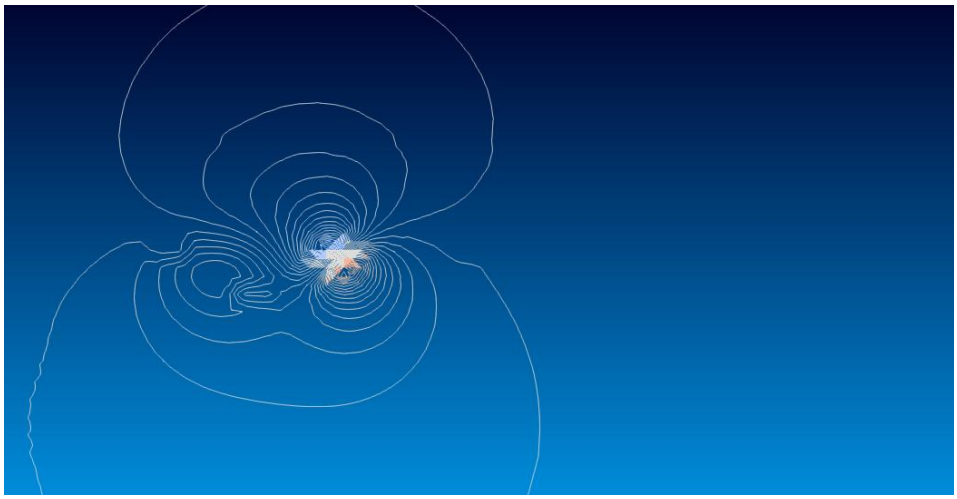
．重合格子のバグ
setFieldsDict での zoneID の順番によって，収束計算の有無が変わる．常に圧力場を見て，確認する必要がある．
例えば，以下のような 5 つの重合格子空間をもつクアッドロータの連成計算を行う場合，



bodyID 1　　// 4 つのペラの重合格子の領域と重なっている．
pera0ID 2
pera1ID 3
pera2ID 4
pera3ID 5
は正しく計算できず，以下のようになる．右は外側の計算空間（background）と収束計算ができていない．



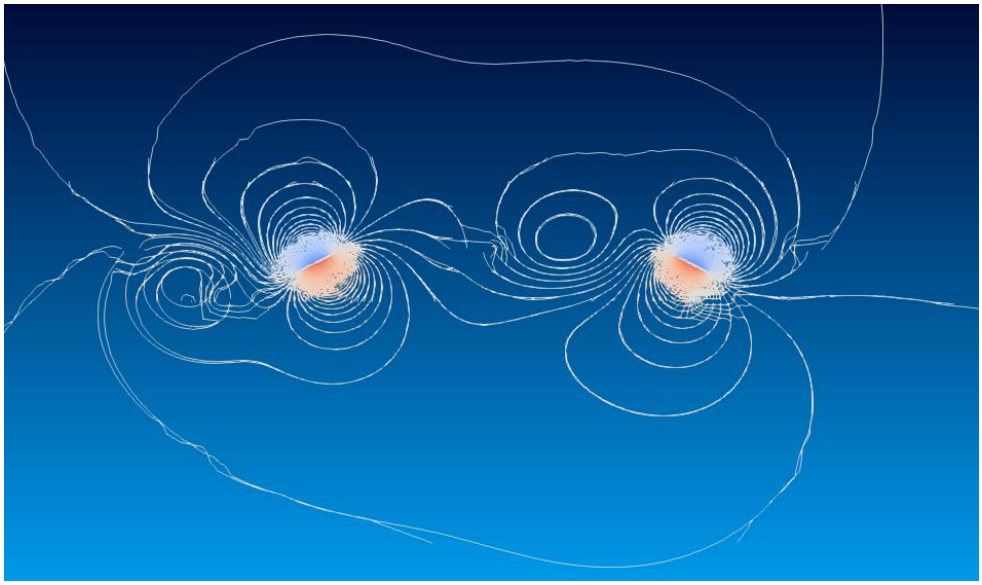
外側の計算空間（background）だけ表示するとこうなる．右側が計算できていない．



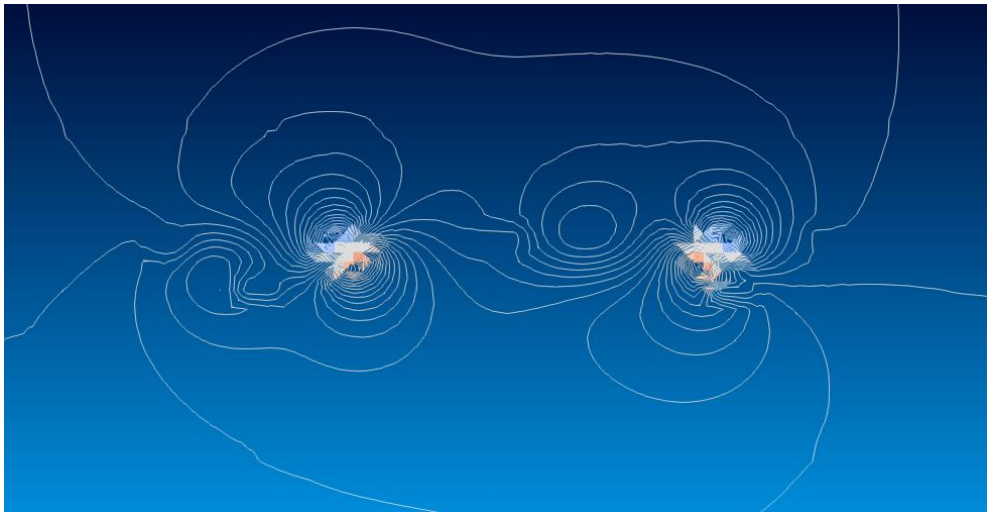
zoneID のナンバリングを変更すると，正しく計算される．

pera0ID 1
pera1ID 2
pera2ID 3
pera3ID 4
bodyID 5　　// 4 つのペラの重合格子の領域と重なっている．

なぜかは不明．恐らくバグ．



background も正しく計算されている.

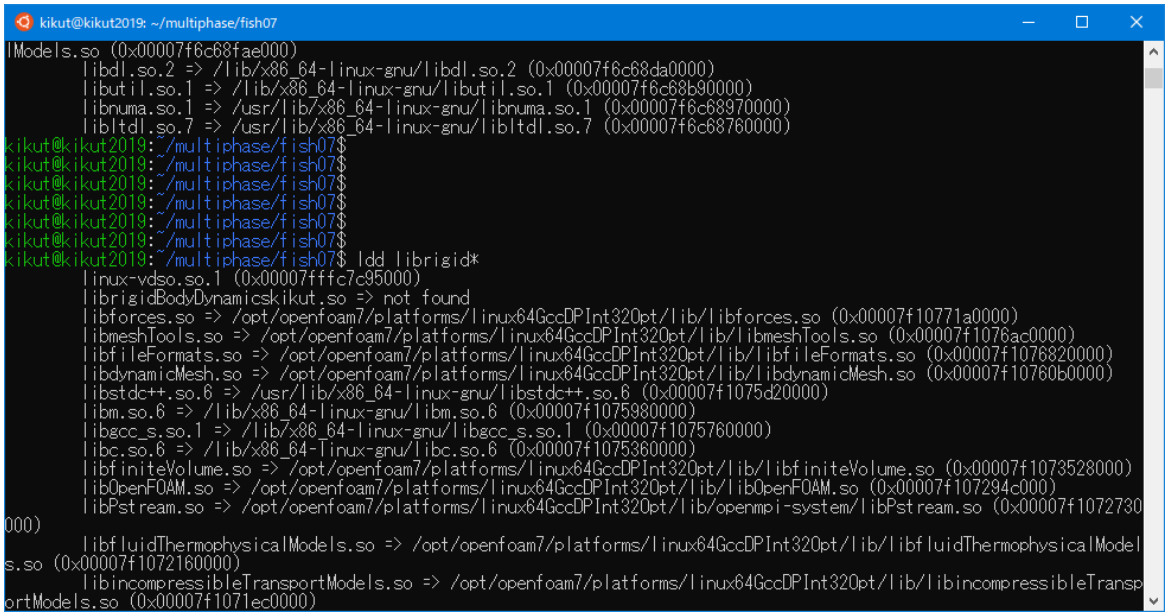


Linux: ライブラリの動的リンクでエラーが出た場合の対処方法（v7 の話）

Selecting dynamicFvMesh dynamicMotionSolverFvMesh
Selecting motion solver: rigidBodyMotion
--> FOAM Warning :
    From function void* Foam::dlOpen(const Foam::fileName&, bool)
    in file POSIX.C at line 1251
    dlopen error : librigidBodyDynamicskikut.so: cannot open shared object file: No such file or directory
--> FOAM Warning :
    From function bool Foam::dlLibraryTable::open(const Foam::fileName&, bool)
    in file db/dynamicLibrary/dlLibraryTable/dlLibraryTable.C at line 105
    could not load "librigidBodyMeshMotionkikut.so"
--> FOAM Warning :
    From function bool Foam::dlLibraryTable::open(const Foam::dictionary&, const Foam::word&, const TablePtr&) [with TablePtr = Foam::HashTable<Foam::autoPtr<Foam::motionSolver> (*)(const Foam::polyMesh&, const Foam::dictionary&), Foam::word, Foam::string::hash>*]
    in file /home/ubuntu/OpenFOAM/OpenFOAM-7/src/OpenFOAM/lnInclude/dlLibraryTableTemplates.C at line 62
    Could not open library "librigidBodyMeshMotionkikut.so"

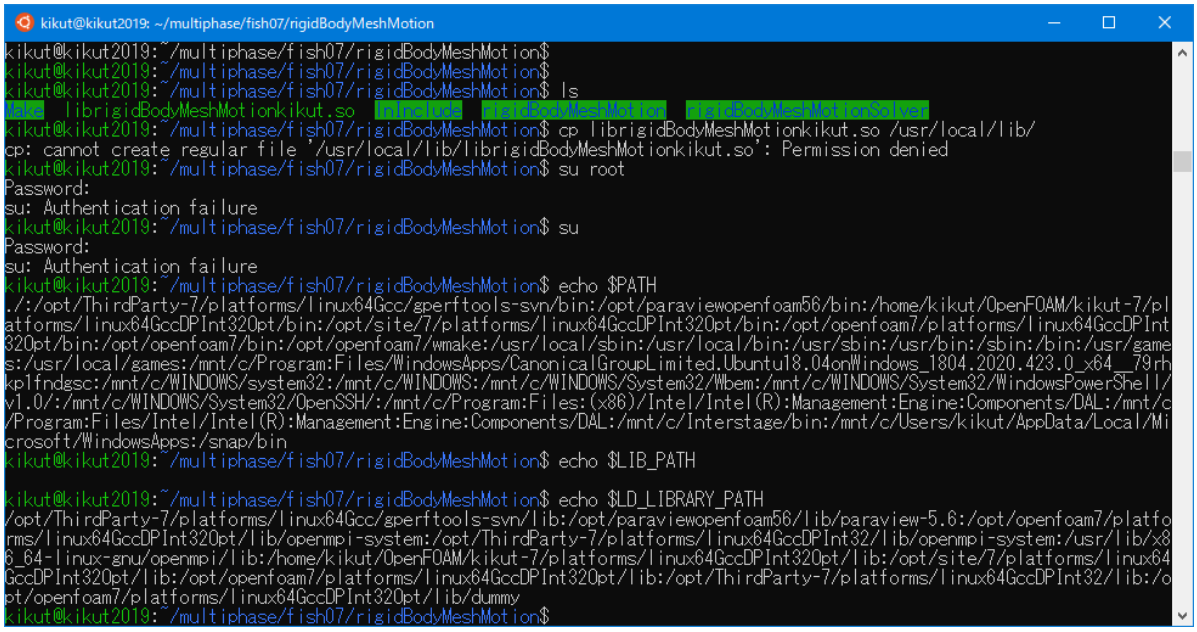
--> FOAM FATAL ERROR:
Unknown solver type rigidBodyMotion

➢ ldd 実行ファイル
とうつと，実行ファイルが参照している dll が示される．



となり，やはり，librigidBodyDynamicskikut.so は not found

➢ echo $LD_LIBRARY_PATH
と打つと，

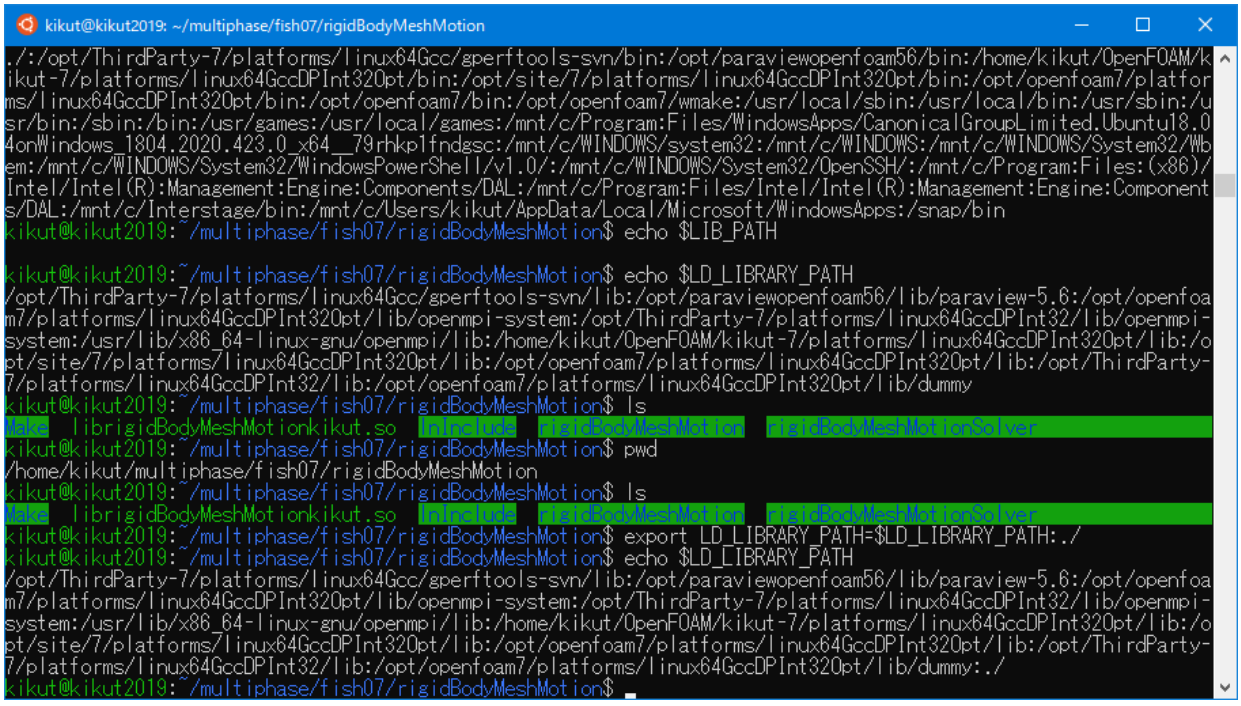


となり，

/home/kikut/OpenFOAM/kikut-7/platfoams/linux64GccDPInt32Opt/lib

にパスが通っていることが分かる．無理やり，カレントにパスを通す．

- export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./



これでもダメ．

結局，

- chmod 644 “libraryfile.so”

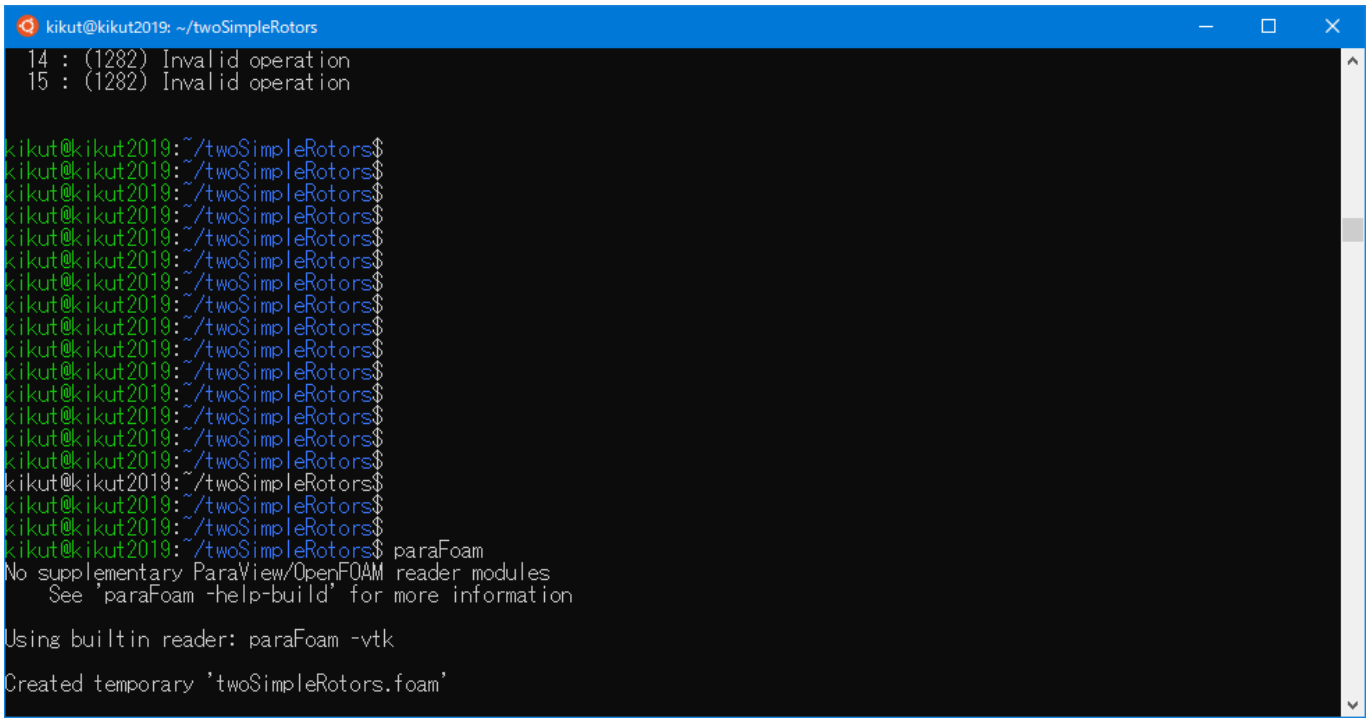
が，ここ

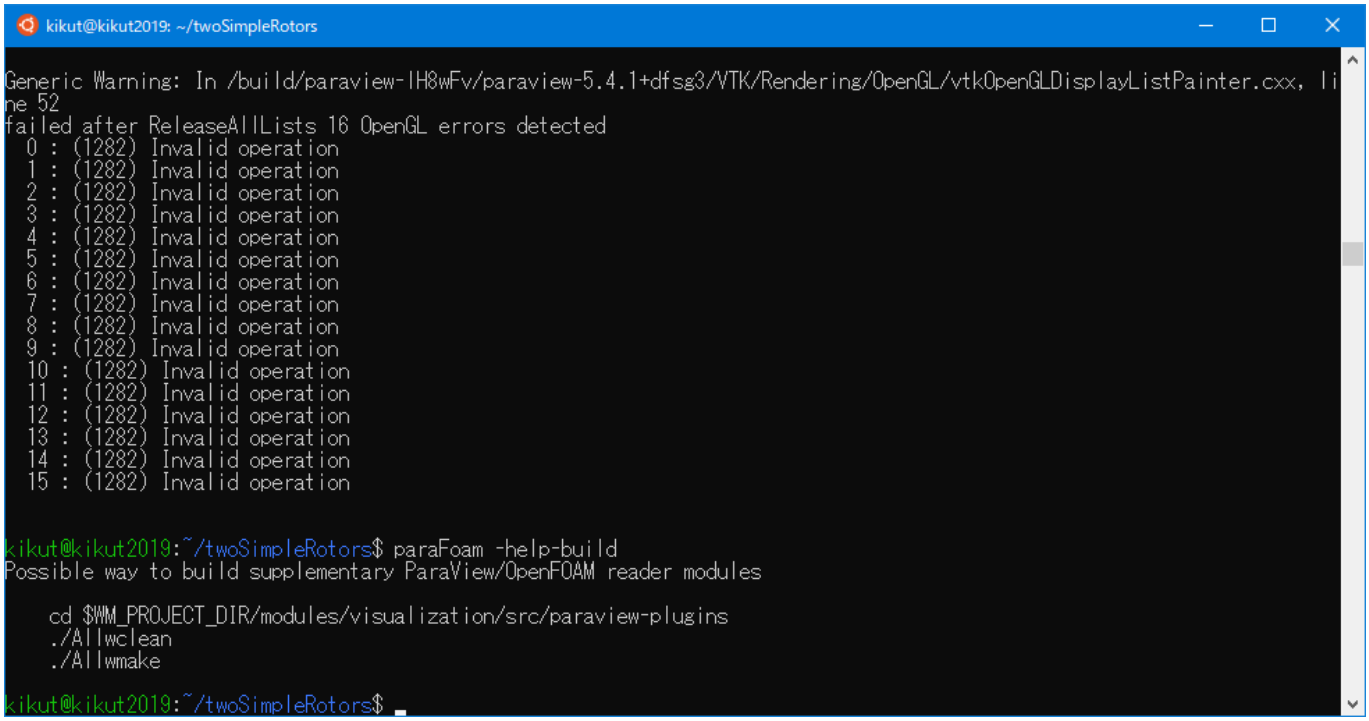
/home/kikut/OpenFOAM/kikut-7/platfoams/linux64GccDPInt32Opt/lib

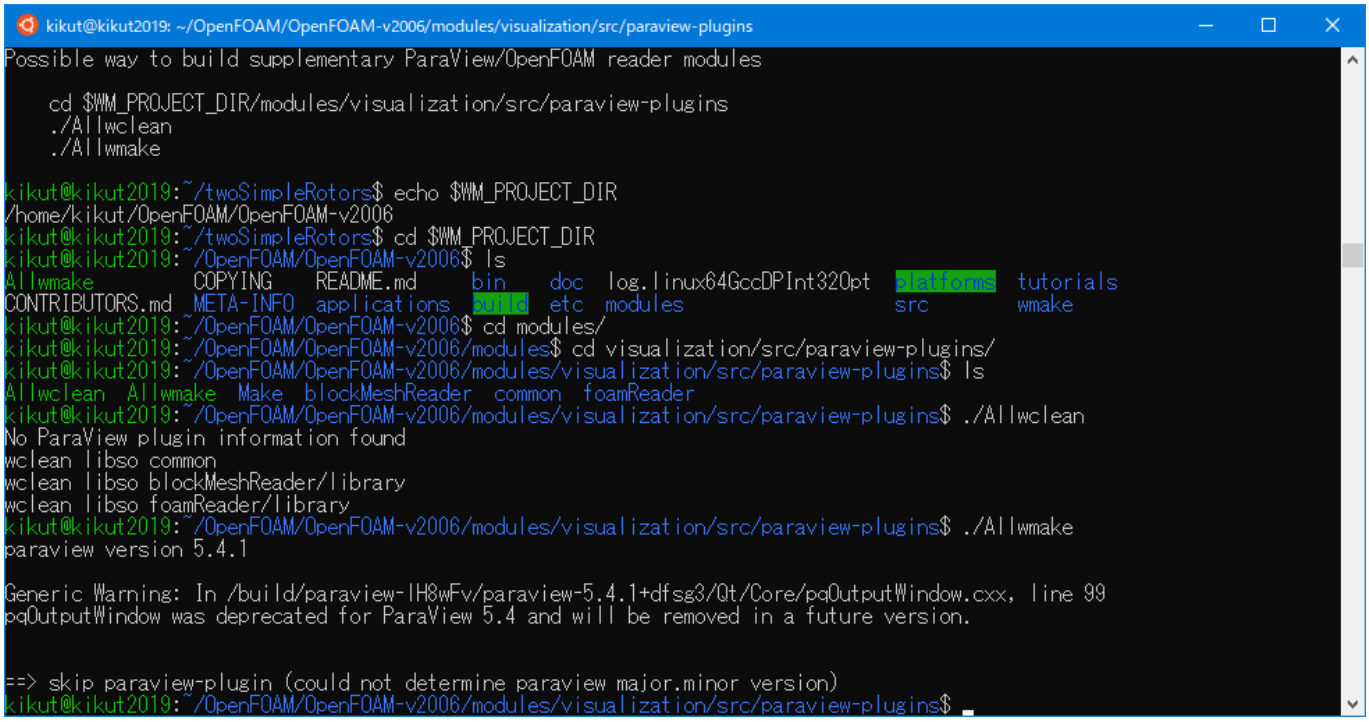
に無いとダメなようだ．本来，ライブラリの場所をしている”-L”オプションなどがあり，読み込めているメッセージがでて，ファイルサイズをみても読み込めているようであるが，ここ以外での作業はいちいち無理だった．

作業ディレクトリごとに作成することはできず，ユーザーごと統一のようだ

paraview error







`sudo apt remove paraview`
sudo apt install /path/to/package.deb
sudo apt install paraview 5.8.0

ライブラリの作成時
wmake libso
のエラーに関して

¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥ThirdParty-v2006¥platforms¥linux64¥gcc-6.3.0¥include¥c++¥6.3.0¥x86_64-pc-linux-gnu¥bits¥os_defines.h
の
features.h
が無いというエラーが出る場合がある．

sudo apt-get update
で更新し，
sudo apt-get install build-essential
でコンパイルツール関係をインストールで改善するような記述があったが，変化はあれど，改善しなかった....

. ver.COM03.7 以前のライブラリにおける双方向連成において，途中から再計算ができなくなる件，
ver.2106 以降仕様変更があり，計算ファイル保存時に，例えば，0.01 秒のデータを保存する場合，

/0.01/uniform//rigidBodyMotionState

が保存されなくなった．これには，各剛体（このライブラリの場合は重心の関節変数）が記載されているがこれがないために，計算途中の剛体変数が再計算時初期化されてしまう（初期値に戻される）．なお，中身はこんな感じ．q は関節の位置姿勢, qDot は速度.

```
/*---------------------------------*- C++ -*----------------------------------*¥
| =========                 |                                                 |
| ¥¥        /   F ield          | OpenFOAM: The Open Source CFD Toolbox           |
```

```
|  ¥¥      /    O peration       | Version:   v2006                                  |
|   ¥¥    /     A nd              | Website:   www.openfoam.com                       |
|    ¥¥/       M anipulation  |                                                       |
¥*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "0.036/uniform";
    object      rigidBodyMotionState;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
q                   9 ( 7.54271e-06 3.54023e-06 -1.65039e-05 0.00124289 -0.00120065 -0.0134153 0.0208358 0.0117894 -
0.0176361 );
qDot                9 ( 0.00088784 0.00924434 -0.000143987 0.067253 -0.0646372 -0.771665 0.424548 2.41249 -1.47427 );
qDdot               9 ( 0.0850193 1.33025 0.0407846 -4.35808 -1.58807 24.7727 -130.303 160.263 36.9835 );
t                   0.036;
deltaT              0.001;
```

仕様変更後（ver.2106～ver.2306 現在）においては,

. 剛体移動（変形ではなく，重心移動）を伴わない計算（CFD, oneway-FSI）は，計算可能．ライブラリコンパイル時，警告はでる．

. 剛体移動（変形ではなく重心移動）を伴う計算に関しては，計算途中から再計算ができない．計算終了まで，止まらずに計算できればシミュレーションはできる．それ以外は，ver.2012 以前を使うのが無難.

. gcc(g++)で，コンパイルができなくて，openFOAM をインストールできない研

おそらく，gcc version11 から，並列処理（MPI 関連）と入出力関係に変更があり，インストール時，Allwmake ができない（コンパイルできないのでメイクできない）状態になっている．ということで，gcc vevrsion10 はコンパイルできるので，こちらをインストールしてコンパイルする．と思いきや，菊池研 FSI ライブラリのファイル入出力にも変更があったようで，最終的に，version9 ならできることを確認した.

- sudo apt install gcc-9
- sudo apt install g++-9

なお，すでにこれより最新の version がインストールされている場合には，以下の設定で切り替える．9 と 13 がインストールされている場合：

- sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 9
- sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-13 13

最後の 9 と 13 は何でもいいが優先順位.

- sudo update-alternatives --config gcc

で，version 9 をデフォルトに選択する．g++も同じ.