Flyingfish Taxing by openFoam v2012

19, August 2021, 更新

混相流（multiphase flow）での一方向流体構造連成（one way FSI）計算を行う．図に示すように，海上の水面でタキシングをするトビウオ周りの流体計算．トビウオは，水面で予め決められた運動を行い，流体力はトビウオに返らない（一方向連成）．流入速度を指定し，トビウオはその一定速度で泳いでいる設定．流体計算には，自由表面（気液境界面移動）を扱う VOF（volume of fluid）法を使っている．液相の境界面の移動に伴って，格子の移動ではなく，格子内の流体の密度比（α）を変更して，境界面運動を表現している．例えば，水のセルは 1（=998.2 [kg/m^3]），空気は 0（=1.293 [kg/m^3]），自由表面上にあるセルの半分が水面下にあるときには，その格子の密度比は 0.5 になる．周りの格子は，トビウオの運動と共に移動，変形する．重合格子（overset mesh）も，適合格子（adaptive mesh）も利用していない．前者は，VOF の流入条件を正しく計算できず（知らないだけで，実はできるかもしれない），後者は格子の移動（ALE 計算）に対応していないようである．よって，極めてシンプルな移動格子の問題である．

以下，計算条件

連続体モデル： ボディは一つ．連続体として変形．体長 L=20cm（計算に反映されないが，質量 m=88g．密度 1000kg/m^3）．

運動：(1)体軸の横波打ち（尾びれ y 方向振り）運動と，(2)体軸の捻り（ロール）運動の式で，トビウオの体表位置が変位する．

$$y(x,t) = A\sin(2\pi(\frac{x}{\lambda} - \frac{t}{T})) \quad where \;\; A = A_0 \frac{x}{L} \qquad (1)$$

尾びれ振り振幅 A は頭部から尾びれに向かうほど大きくなり，波長 λ はここでは体長 L に等しく，周期 T は周波数 f=40[Hz]の逆数である．体長が長くなるが，微小としてここでは無視している．x はトビウオ座標系体軸で，頭部が x=0 である．

$$\theta(x,t) = B\sin(2\pi ft + \phi) \quad where \;\; B = B_0 \frac{x}{L} \qquad (2)$$

捻り運動の振幅 B は頭部から尾びれに向かうほど大きくなり，周波数 f，位相 ϕ である．流入速度の設定により，トビウオは，10m/s で泳いでいる設定．

10m/s で遊泳しているトビウオ周りのレイノルズ数は，

水中 $\mathrm{Re} = \frac{UL}{\nu} = \frac{10*0.2}{1.0*10^{-6}} = 2.0*10^6$

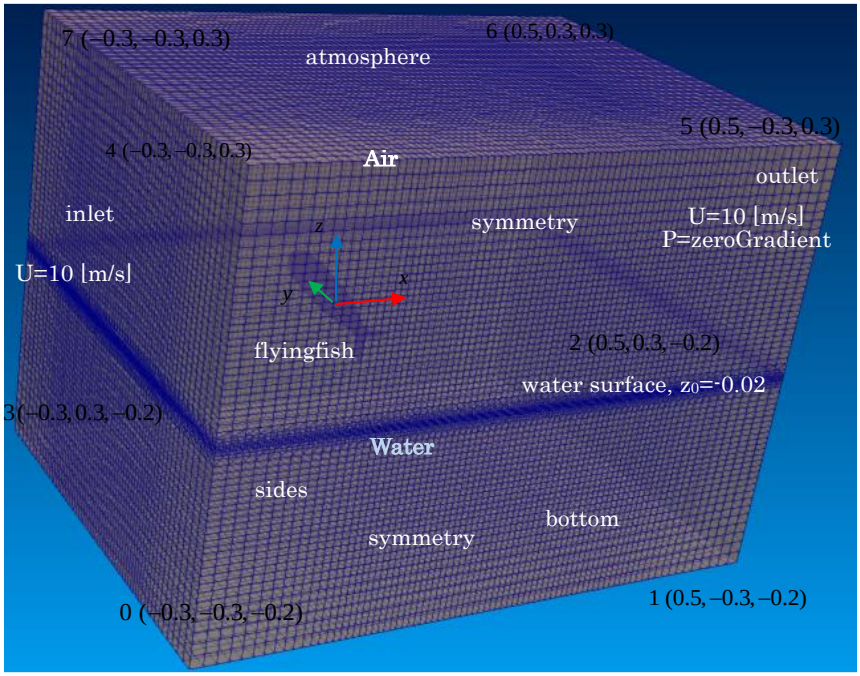
ストローハル数は，

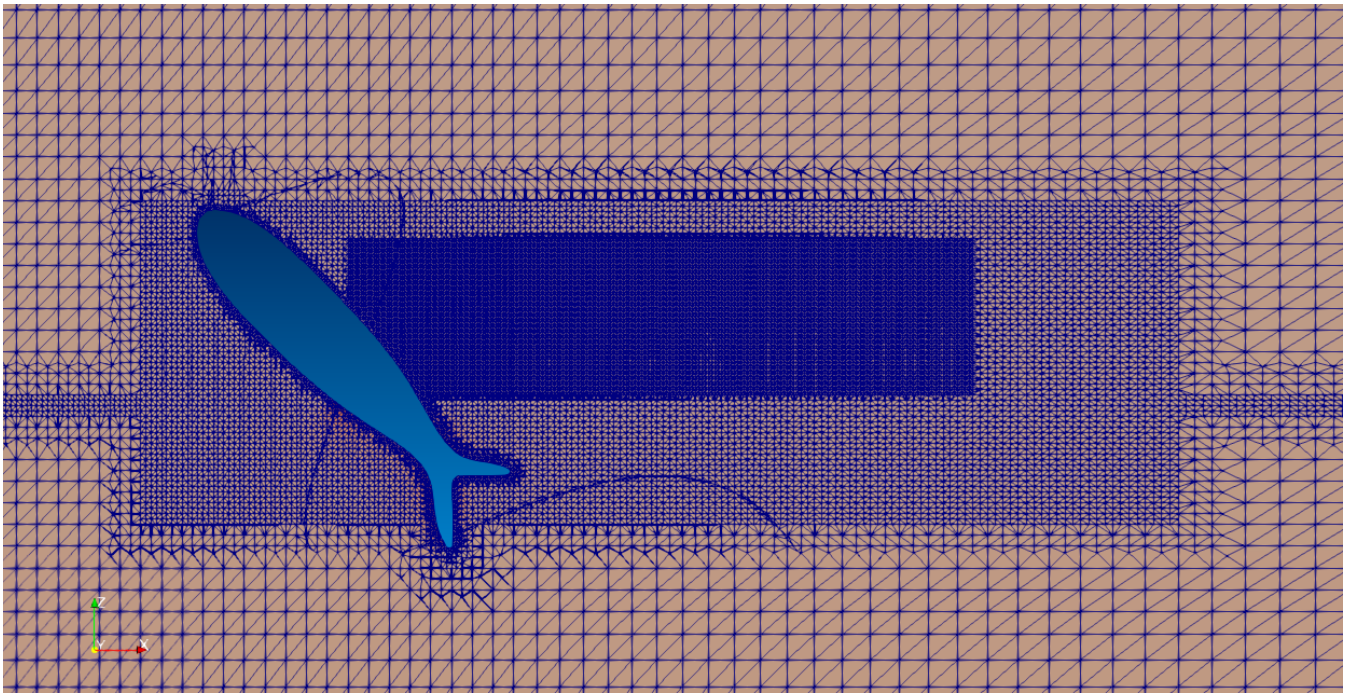
$\mathrm{Re} = \frac{fL}{U} = \frac{40*0.2}{10} = 0.8$

ウェーバー数は，

$\mathrm{Re} = \frac{\rho LU^2}{\sigma} = \frac{1000*0.2*10^2}{0.072} = 2.7*10^5$

である．乱流であるので，k-ωで計算してみる．以下，ARM を使っていないため，水面と尾びれ回りを snappy で level2（1/2 の 1/2）に，尾びれ後方の空間を level3（1/2 の 1/2 の 1/2）にしてある．全体の格子サイズが概ね 1cm 弱なので，水面は 1/4cm，尾びれ後方は 1/8cm．これでも ϕ 1mm 程度の水滴を表現するには粗い．水滴の予備計算では，さらに 1/2 に細分化して正しくクラウンを表現できていた．予備計算により，クラウンの発生しそうな空間を level3 で細分化してある．これで，だいたい 1 データ 500Mbyte.





境界条件

	Type	alpha.water	U 速度	p_rgh 圧力と水圧
atmosphere	patch	type inletOutlet; inletValue uniform 0; value uniform 0;	type pressureInletOutletVelocity; tangentialVelocity niform (10 0 0); value uniform (0 0 0);	**type totalPressure;** **p0 uniform 0; 初期値？**
inlet	patch	type fisedValue; value uniform 0;	type fixedValue; value uniform (10 0 0);	type fixedFluxPressure; value uniform 0
outlet	patch	type variableHeightFlowRate; lowerBound 0; upperBound 1; value uniform 0;	type outletPhaseMeanVelocity; alpha alpha.water; Umean 10; 平均流速 value $internalField;	type zeroGradient
flyingfish	wall	zeroGradient	type movingWallVelocity; value uniform (0 0 0);	type fixedFluxPressure; value uniform 0;

	k	nut	omega	pointDisplacement

atmosphere	type inletOutlet; inletValue uniform 0.00015; value uniform 0.00015;	type zeroGradient;	type inletOutlet; inletValue uniform 3.06; value uniform 3.06;	type fixedValue; value uniform (0 0 0)
inlet	type fixedValue; value uniform 0.00015;	type fixedValue; value uniform 4.9e-05;	type fixedValue; value uniform 3.06;	type fixedValue; value uniform (0 0 0)
outlet	type inletOutlet; inletValue uniform 0.00015; value uniform 0.00015;	type zeroGradient;	type inletOutlet; inletValue uniform 3.06; value uniform 3.06;	type fixedValue; value uniform (0 0 0)
sides, bottom				type fixedValue; value uniform (0 0 0)
flyingfish	type kqRWallFunction; value uniform 0.00015;	type nutkRoughWallFunction; Ks uniform 100e-6; Cs uniform 0.5; value uniform 4.9e-05;	type omegaWallFunction; blending binomial2; value uniform 3.06;	type calculated; value uniform (0 0 0);

symmetry 設定の底面と側面の境界は，include から読み込んでいる．

1. ライブラリの作成

COM03 のライブラリ使い，以下のファイルのみ変更している．

．rigidBodyDynamics/rigidBodyMotion/rigidBodyMotion.H

ヘッダ．

```
#ifndef rigidBodyMotion_H
#define rigidBodyMotion_H

#include "rigidBodyModel.H"
#include "rigidBodyModelState.H"
#include "pointField.H"
#include "Switch.H"

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

namespace Foam
{

// Forward declarations
class Time;

namespace RBD
{

// Forward declarations
class rigidBodySolver;

/*---------------------------------------------------------------------------*¥
                        Class rigidBodyMotion Declaration
¥*---------------------------------------------------------------------------*/

class rigidBodyMotion
:
    public rigidBodyModel
{
    friend class rigidBodySolver;

    // Private data

        //- Motion state data object
        rigidBodyModelState motionState_;

        //- Motion state data object for previous time-step
        rigidBodyModelState motionState0_;
```

```
        //- Initial transform for external forces to the bodies reference frame
        List<spatialTransform> X00_;

        //- Acceleration relaxation coefficient
        scalar aRelax_;

        //- Acceleration damping coefficient (for steady-state simulations)
        scalar aDamp_;

        //- Switch to turn reporting of motion data on and off
        Switch report_;

        //- Switch to turn FSI calculation on and off   oneway FSI のフラグ．以下，constant/dynamicMeshDict から読み込む
        Switch FSI_;

          scalar      fishLength_;                  // body length            トビウオ体長
          vector      fishCenter_;                  // fish center position (x y z) [m]   トビウオ中心
          scalar      pitchAttitude_;               // fish pitch attitude [rad]          ピッチ姿勢
          scalar      motionFreq_;                  // motion frequency [Hz]              尾びれ振り周波数
          scalar      waveLength_;                  // wave length [m]                    尾びれの波長
          scalar      amplitude_;                   // wave motion amplitude [m]          尾びれの振幅
          vector      rollMotion_;          // roll motion (amplitude [rad], freq [hz], phase [rad])   ロール運動パラメータ
          scalar      relaxation_time_;     // relaxation time for stable calculation [s]   緩和時間．1 周期程度

        //- Motion solver
        autoPtr<rigidBodySolver> solver_;
```

. . .

．rigidBodyDynamics/rigidBodyMotion/rigidBodyMotion.C

プログラム．

```
Foam::RBD::rigidBodyMotion::rigidBodyMotion(const Time& time)
:
    rigidBodyModel(time),
    motionState_(*this),
    motionState0_(*this),
    aRelax_(1.0),
    aDamp_(1.0),
    report_(false),
    FSI_(true),
    solver_(nullptr)
{}

Foam::RBD::rigidBodyMotion::rigidBodyMotion
(
    const Time& time,
    const dictionary& dict
)
:
    rigidBodyModel(time, dict),
    motionState_(*this, dict),
    motionState0_(motionState_),
    X00_(X0_.size()),
    aRelax_(dict.getOrDefault<scalar>("accelerationRelaxation", 1)),
    aDamp_(dict.getOrDefault<scalar>("accelerationDamping", 1)),
    report_(dict.getOrDefault<Switch>("report", false)),
    FSI_(dict.getOrDefault<Switch>("FSI", true)),
    solver_(rigidBodySolver::New(*this, dict.subDict("solver")))
{
```

```
    if (dict.found("g"))
    {
        g() = dict.get<vector>("g");
    }

    initialize();

    Iteration_number_for_MB_ = dict.getOrDefault<int>("Iteration_number_for_MB", 1);

    if( FSI() ){                        // 一方向連成，or 双方向連成
    }else{
        dict.readEntry("fishLength", fishLength_);     // パラメータ読み込み
        dict.readEntry("fishCenter", fishCenter_);
        dict.readEntry("pitchAttitude", pitchAttitude_);

        dict.readEntry("motionFreq", motionFreq_);
        dict.readEntry("waveLength", waveLength_);
        dict.readEntry("amplitude", amplitude_);

        dict.readEntry("rollMotion", rollMotion_);

        dict.readEntry("relaxation_time", relaxation_time_);
    }
}


Foam::RBD::rigidBodyMotion::rigidBodyMotion
(
    const Time& time,
    const dictionary& dict,
    const dictionary& stateDict
)
:
    rigidBodyModel(time, dict),
    motionState_(*this, stateDict),
    motionState0_(motionState_),
    X00_(X0_.size()),
    aRelax_(dict.getOrDefault<scalar>("accelerationRelaxation", 1)),
    aDamp_(dict.getOrDefault<scalar>("accelerationDamping", 1)),
    report_(dict.getOrDefault<Switch>("report", false)),
    FSI_(dict.getOrDefault<Switch>("FSI", true)),
    solver_(rigidBodySolver::New(*this, dict.subDict("solver")))
{
    if (dict.found("g"))
    {
        g() = dict.get<vector>("g");
    }

    initialize();

    Iteration_number_for_MB_ = dict.getOrDefault<int>("Iteration_number_for_MB", 1);

    if( FSI() ){
    }else{
        dict.readEntry("fishLength", fishLength_);
        dict.readEntry("fishCenter", fishCenter_);
        dict.readEntry("pitchAttitude", pitchAttitude_);

        dict.readEntry("motionFreq", motionFreq_);
        dict.readEntry("waveLength", waveLength_);
        dict.readEntry("amplitude", amplitude_);
```

```
            dict.readEntry("rollMotion", rollMotion_);

            dict.readEntry("relaxation_time", relaxation_time_);
        }
}
...
Foam::tmp<Foam::pointField> Foam::RBD::rigidBodyMotion::transformPoints //  一剛体用ルーチン
(
        const label bodyID,
        const scalarField& weight,
        const pointField& initialPoints
) const
{
        // Calculate the transform from the initial state in the global frame
        // to the current state in the global frame
        spatialTransform X(X0(bodyID).inv() & X00(bodyID));

        // Calculate the septernion equivalent of the transformation for 'slerp'
        // interpolation
        septernion s(X);

        tmp<pointField> tpoints(new pointField(initialPoints));
        pointField& points = tpoints.ref();

   if( FSI() ){             // *** FSI routine ***  双方向連成ルーチン
Info << " rigidBodyMotion::transformPoints FSI " << endl;
        forAll(points, i)
        {
            // Move non-stationary points
            if (weight[i] > SMALL)
            {
                // Use solid-body motion where weight = 1
                if (weight[i] > 1 - SMALL)
                {
                    points[i] = X.transformPoint(initialPoints[i]);
                }
                // Slerp septernion interpolation
                else
                {
                    points[i] =
                        slerp(septernion::I, s, weight[i])
                       .transformPoint(initialPoints[i]);
                }
            }
        }
   }else{                   // *** oneway FSI routine ***    一方向連成ルーチン
Info << " rigidBodyMotion::transformPoints oneway-FSI " << endl;
            const scalar t = state().t();           //  現在時間

            vector      pos;                        //  位置ベクトル定義
            scalar      droll, dy;                  //  ロール角度，y 変位
            Tensor<scalar> Rx, Ry;                  //  回転行列

            //  ピッチ姿勢行列．行と列に注意
            Ry.xx() = cos(pitchAttitude_);      Ry.xy() = 0.0;          Ry.xz() = sin(pitchAttitude_);      // waring:: inversed
            Ry.yx() = 0.0;                                  Ry.yy() = 1.0;          Ry.yz() = 0.0;
            Ry.zx() =-sin(pitchAttitude_);      Ry.zy() = 0.0;          Ry.zz() = cos(pitchAttitude_);

        forAll(points, i){  //  全格子ルーチン
//          if (weight[i] > SMALL){             //
```

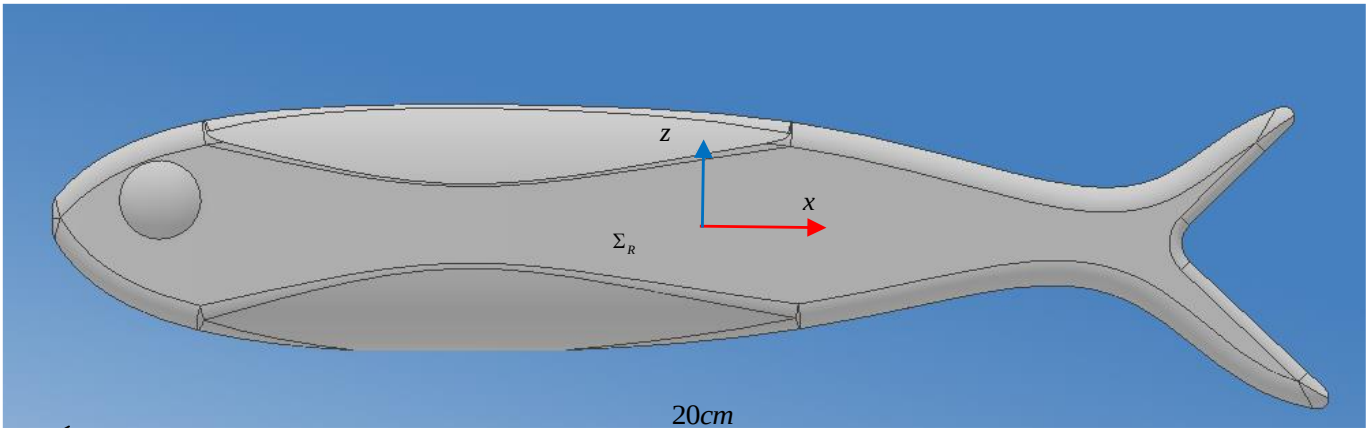
```
            pos = ( Ry.T() & initialPoints[i] ) - fishCenter_;    // 魚中心を原点にした魚座標系に並進変換
            if( mag( pos ) < fishLength_ ){        // 魚の周りの格子なら，y 変位を計算
                if( pos.x() < (-fishLength_/2.0) )                // 魚の頭より前なら
                        dy = 0.0;
                else if( pos.x() < (fishLength_/2.0) )            // 魚部分なら
                        dy = amplitude_ / fishLength_ * ( pos.x() + (fishLength_/2.0) ) * sin( 2.0 * PAI * ( ( pos.x() + (fishLength_/2.0) ) / waveLength_ - motionFreq_ * t ) );
                else                                              // 魚より後ろなら
                        dy = amplitude_ * sin( 2.0 * PAI * ( fishLength_ / waveLength_ - motionFreq_ * t ) )
                            + ( amplitude_ / fishLength_ * sin( 2.0 * PAI * ( fishLength_ / waveLength_ - motionFreq_ * t ) )
                              + amplitude_ * cos( 2.0 * PAI * ( fishLength_ / waveLength_ - motionFreq_ * t ) ) * 2.0 * PAI / waveLength_ ) * ( pos.x() - (fishLength_/2.0) );

                if( (fishLength_/2.0) < mag( pos ) )              dy *= sqr( 1.0 - ( mag( pos ) - (fishLength_/2.0) ) / (fishLength_/2.0) );   // 位置緩和
                if( t < relaxation_time_ )                        dy *= ( t / relaxation_time_ );       // 時間緩和

                pos.y() += dy;

              // ロール回転ルーチン
                if( pos.x() < (-fishLength_/2.0) )                droll = 0.0;
                else if(pos.x() < (fishLength_/2.0) )             droll = rollMotion_[0] / fishLength_ * ( pos.x() + (fishLength_/2.0) ) * sin( 2.0*PAI*rollMotion_[1]* t + rollMotion_[2] );
                else                                              droll = rollMotion_[0] * sin( 2.0*PAI*rollMotion_[1]* t + rollMotion_[2] );

                if( (fishLength_/2.0) < mag( pos ) )              droll *= ( 1.0 - ( mag( pos ) - (fishLength_/2.0) ) / (fishLength_/2.0) );
                if( t < relaxation_time_ )                        droll *= ( t / relaxation_time_ );

                Rx.xx() = 1.0;      Rx.xy() = 0.0;                  Rx.xz() = 0.0;                  // waring:: inversed
                Rx.yx() = 0.0;      Rx.yy() = cos(droll);   Rx.yz() =-sin(droll);
                Rx.zx() = 0.0;      Rx.zy() = sin(droll);   Rx.zz() = cos(droll);

                points[i] = Ry & ( ( Rx & pos ) + fishCenter_ );
            }else{
                points[i] = initialPoints[i];
              }
//          }
        }
    }
    return tpoints;
}
```

注：その後，格子移動範囲を 1.5 倍に広げた．

2. CAD モデルの生成

CAD でトビウオモデルを作図する（n 剛体用に分解しておく必要はない）．stl ファイルにするときの設定に注意する（メートル，ASCII 設定）．CAD の原点が openFoam の原点となる．一方向連成なので，質量も慣性も必要ない．流体からの反力は計算できるが，トビウオ自体には何も反映されない．ピッチ角 45deg でタキシングを行うようにアセンブリした後，以下のディレクトリにファイルを保存しておく．

constant/triSurface/*.stl



3. 境界条件の設定

各種境界条件の設定法は，以下を参照.

https://www.cfd-online.com/Wiki/Turbulence_free-stream_boundary_conditions

. k

Turbulent energy [m^2/s^2]は

$$k = \frac{3}{2}(UI)^2$$

U: the mean flow velocity, I: the turbulence intensity

ここでは，U=10 [m/s], I=0.001 (=0.1%)とした．k=0.00015.

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

internalField   uniform 0.015;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    inletSide
    {
        type            fixedValue;
        value           $internalField;
    }

    outletSide
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }

    atmosphere
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }

    flyingfish
    {
        type            kqRWallFunction;
        value           $internalField;
    }
}
```

kqRWallFunction:
https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-wall-turbulence-kqRWallFunction.html

.nut
Modified turbulent viscosity [m^2/s]

$$\tilde{\nu} = \sqrt{\frac{3}{2}}(UIl)$$

U: the mean flow velocity, I: the turbulence intensity, l: the turbulent length scale
ここでは，U=10 [m/s], I=0.001 [=0.1%], l は経験的に求めるが，例えば，代表長さの 7%とか．ここでは，トビウオを 100 分割と考えて，2%とし，l=0.2*0.02=0.004 としてみた．4mm のメッシュで表現できない乱流が考慮されることになるのか．．．．？

$\tilde{\nu} < \frac{\nu}{2}$ は，問題が起こる場合があるとか，$\tilde{\nu} = 5\nu$ が便利であるとかの記述があるが，よく分からず．
nut=4.9*10^-5
となるが，$\nu_{水}$＝10^-6，$\nu_{空気}$=1.5*10^-5 であり，オーダーはだいたい合っているか？

```
dimensions      [0 2 -1 0 0 0];

internalField   uniform 4.9e-05;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    inlet
    {
        type            fixedValue;
        value           $internalField;
    }

    outlet
    {
        type            zeroGradient;
    }

    atmosphere
    {
        type            zeroGradient;
    }

    flyingfish
    {
        type            nutkRoughWallFunction;    // 粗い壁の関数．ザラザラの紙やすりみたいな表面？
        Ks               uniform 100e-6;
        Cs               uniform 0.5;
        value           $internalField;
    }
}
```

nutkWallFunction：
https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-wall-turbulence-nutkWallFunction.html

nutkLowReWallFunction：
https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-wall-turbulence-nutLowReWallFunction.html

nutkRoughWallFunction：
https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-wall-turbulence-nutkRoughWallFunction.html

nutWallFunction:
https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-wall-turbulence-nutWallFunction.html

.omega

もっともシンプルな ω [/s]として，

$$\omega = \frac{\sqrt{k}}{l}$$

l: the turbulent length scale

を用いる．上記より， ω=3.06

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

internalField   uniform 8.7;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    inletSide
    {
        type            fixedValue;
        value           $internalField;
    }

    outletSide
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }

    atmosphere
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }

    flyingfish
    {
        type            omegaWallFunction;
        blending        binomial2; // tutorial では，blended true となっているが，警告が出るので修正してある．
        value           $internalField;
    }
}
```

omegaWallFunction に関しては以下参照：

https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-wall-turbulence-omegaWallFunction.html

4. 運動学に基づく格子移動の設定

constant/dynamicMeshDict ファイルを用意する．

```
dynamicFvMesh       dynamicMotionSolverFvMesh;  // ここでは，重合ではなく，移動格子を選択

motionSolverLibs    (rigidBodyMeshMotionCOM);   // COM ver3 以降を選択
motionSolver        rigidBodyMotion;

report              off;
g                   ( 0 0 -9.8065 );

solver
{
    type NewmarkCOM;
//      gamma     0.75;       // Velocity integration coefficient:          "gamma > 0.5" is unconditionary stable, but low accuracy.
```

```
//      beta      0.390625;      // Position integration coefficient:      beta = (gamma+0.5)^2/4
}

Iteration_number_for_MB          1;

// OutputFiles             (flyingfishbody);        // save the center of rotation.

golocalCoordinateSystem          (1 0 0    0 1 0    0 0 1);

bodies
{
// 剛体をひとつ定義．複数定義すると，overload された異なる関数が呼び出されるので注意．物性値は使っていない．
    flyingfishbody
    {
        type                rigidBody;
        parent              root;
        mass                 0.090;
        inertia             (3940e-9 0 0    4210e-9 0    2140e-9);
        centreOfMass        (-0.014 0.014 0);                          /* global coordinate system */
        transform           $golocalCoordinateSystem $centreOfMass;
        joint {
            type                composite;
            joints (
                {    type Pxyz;            }
                {    type Rxyz;            }
            );
        }
        patches             (flyingfish);
        innerDistance       0;
        outerDistance       101;
    }
}

restraints
{
}

FSI          false;       // false にすると，剛体の動力学計算がすべて無視され，連続体の一方向計算となる．

// ******* flyingfish parameters *******    魚の運動パラメータ
fishLength  0.2;          // fish body length [m]
fishCenter  (0 0 0);      // fish center position (x, y, z) [m]
pitchAttitude             0.785398;  // fish pitch attitude [rad]

motionFreq40;             // wave motion frequency [Hz]
waveLength                0.2;          // wave length for wave motion [m]
amplitude   0.02;         // wave motion maplitude[m]

rollMotion (0.261799 40 1.570796);              // roll motion parameters (amplitude [rad], freq [Hz], phase [rad] )

relaxation_time 0.025; // relaxation time for stable calculation
```

5.　解析用のモーメント計算用の点の記述

system/controlDict ファイルに流体力によるモーメントを計算する点をいくつか設定しておく．重心が分かれば重心がよい．

```
functions
{
    forces
    {
        type                forces;
        libs                (forcesCOM);
```

```
        patches         (flyingfish);
        rhoInf          998.8;
        log             off;
        writeControl    timeStep;       // この設定，うまくいっていない．
        writeInterval   10;
        CofR            (0.01414 0 0.01414);    // 重心位置
    }
}
```

6. 実行

三ケース示す．

Case A: 重合格子も適合格子も使わず，尾びれ周りだけ細分化（2mm 程度），乱流条件 k-ω， ν^適当．

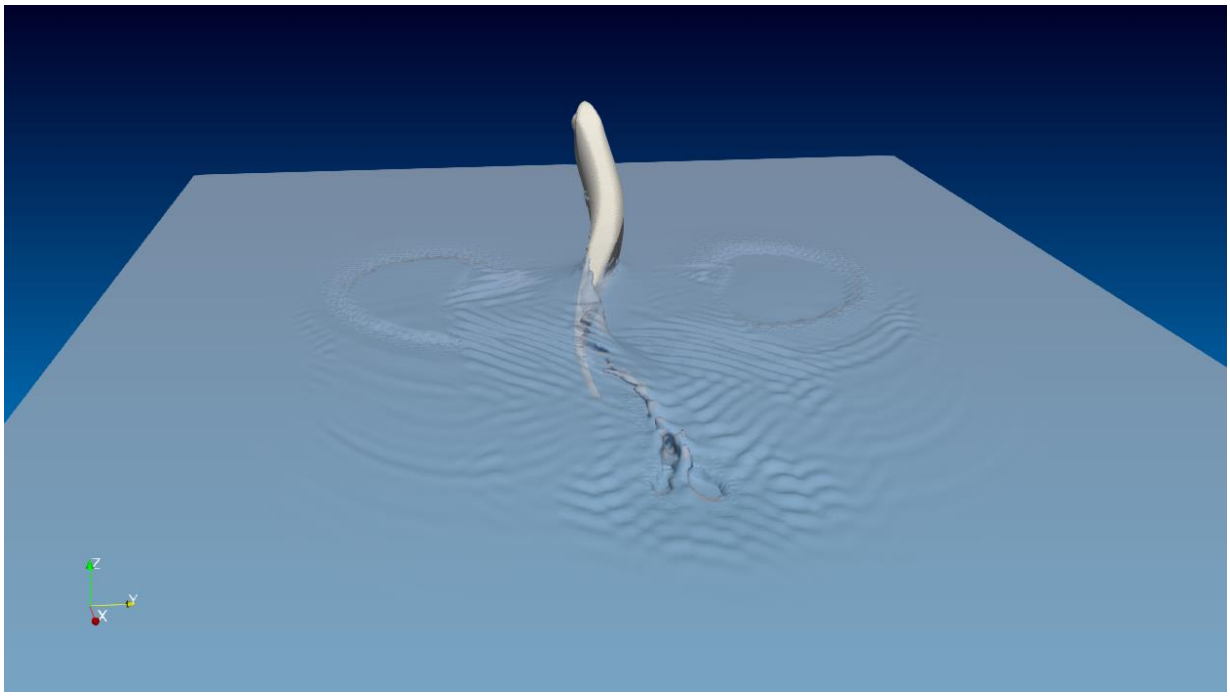
Case B: 重合格子も適合格子も使わず，尾びれ後方も細分化（1mm 程度），乱流条件 k-ω， ν^修正．（本プログラム）

Case C: 適合格子で液面を細分化（1mm 程度）

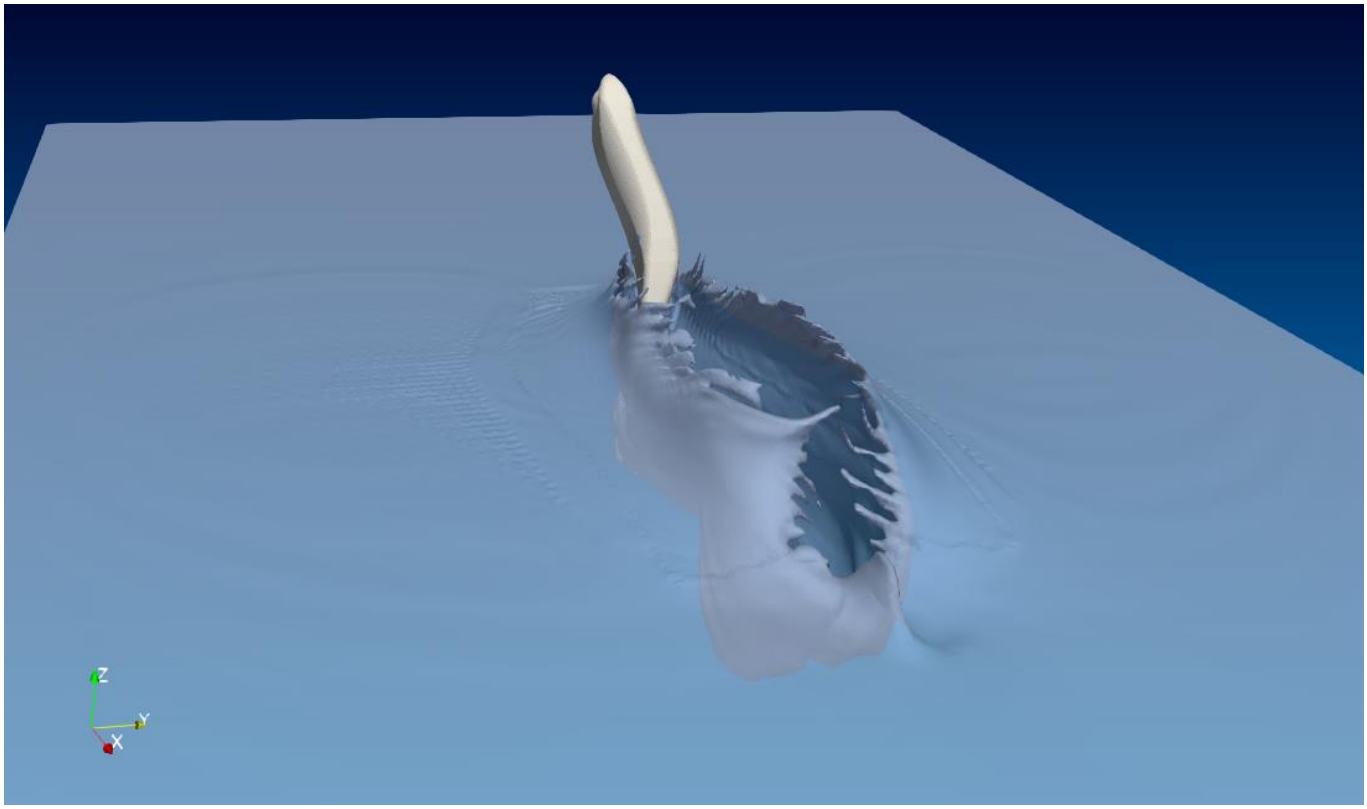
Case A は，境界条件の設定が間違っていたかも．．．？　力学は正しいが，格子が粗すぎて，クラウンどころか，液面の形状なども表現できていない．なお，計算途中の境界（おそらくトビウオ境界）のωのレンジが 10^50 などのオーダーで大きくなっていた．原因は分からず．一般論として，乱流のエネルギーが大きくなっているので，尾びれ周りの粘性が大きくなると思われるが，k の値は大きくなっていないようだ．ということは，乱流長さスケールの l が小さく見積もられている？

Case B は，本プログラム．そこそこクラウンも表現でき，よい解のように思われる．要検証．細分化はあと一段階行うと，水面のスラップ時の水しぶきなども表現できるが，一データ 1Gbyte を越えたため諦めた．上記のωのレンジは一周期を越えると，10^2 のオーダーとなっていた．気になるが原因は分からず．

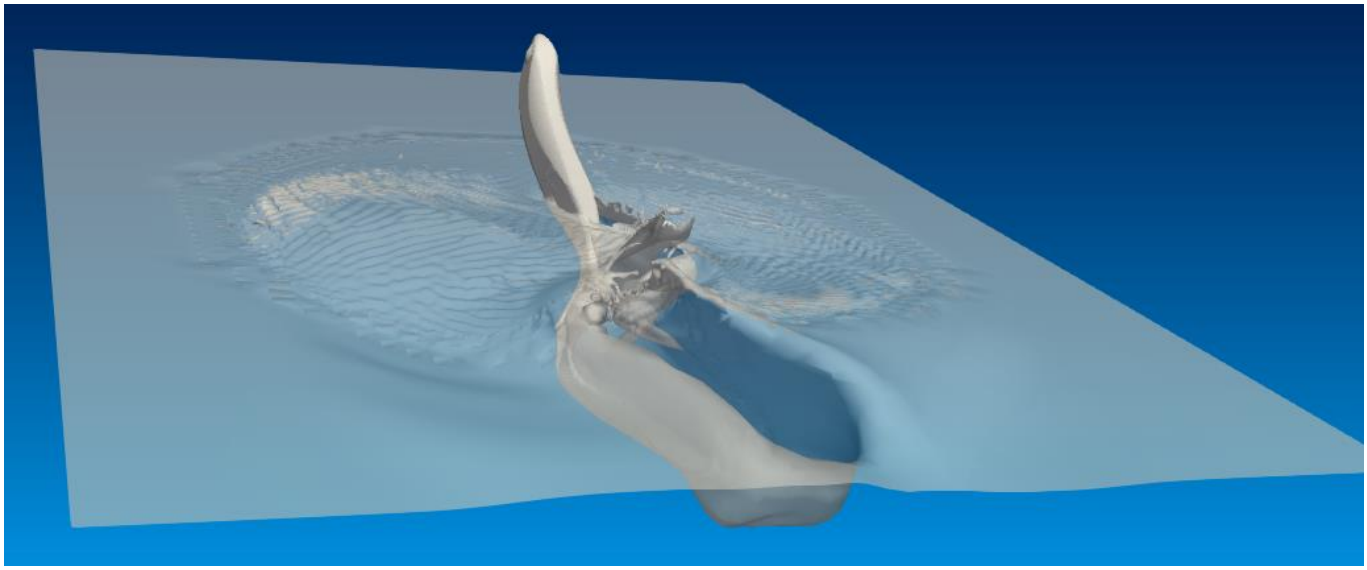
Case C は，そこそこ実際に近い状況で，適合格子なのでクラウンも表現できそうだが，力学が正しくない．適合格子で分割した格子，及び移動した格子の力学が正しく計算されていないように思われる．要検証．格子が移動すると，流体の物理量も移動してしまうようだ．格子が移動しても，水は移動してはいけない．なお，トビウオ周りのみを適合格子で細分化しているため，その外側の液面はなまっているのが分かる．



Case A: 重合格子も適合格子も使わず，尾びれ周りだけ細分化（2mm 程度）．乱流設定がうまくいっていないか？



Case B: 重合格子も適合格子も使わず，尾びれ後方だけさらに細分化（1mm 程度），乱流境界条件も修正


Case C: 適合格子で液面を細分化（1mm 程度）

なお, 以下は, 0.0335 秒後. すでに細分化した領域を水しぶきが越えていることが分かる. 単純に考えると, 10m/s なので, 33.5cm の長さということになる．後方にもっと細分化したいが，70cm の範囲を見ようとすると， 1 ファイル 1Gbyte．なお，5cm 舞い上がった水しぶきが水面に落下する時間は，0.1 秒.