Flyingfish Takeoff by openFoam v2012

18, May 2022, 更新

混相流（multiphase flow）での流体構造連成（FSI）計算を行う．水中から水面に対して垂直に離水するトビウオ周りの流体計算．トビウオの周りは移動重合格子（overset mesh）で，トビウオの運動と共に移動・変形する．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．構造計算は，マルチボディによる動力学計算．結合はシリアルリンクと同じだが，伸縮はする．

以下，計算条件方法．

マルチボディモデル： ボディは 12 分割でシリアルに結合．ボディ間は単純なバネダンパ．体長 L=20cm（質量 m=88g．密度 1000kg/m^3）．体軸を x として，ヨー運動（z 軸回転）するため，ボディそれぞれの z 軸回り慣性モーメントが大きく異ならないように分割する．

遊泳運動：(1)体軸の横波打ち（尾びれ y 方向振り：z 回転）運動と，(2)体軸の捻り（ロール：x 回転）運動の式で，重合格子内のトビウオの体表位置（トビウオ境界）が変位する．(1)の運動は，Carangiform の近似として後述する式を用いる．尾びれ振り振幅が，体軸方向（頭部から尾びれに向かう）ほど二次関数的に大きくなる．波長はここでは体長に等しく，周波数は f=20 [Hz] である． (2)の運動では，捻り運動の振幅は体軸方向ほど線形に大きくなり，周波数 f，位相 ϕ である．ただし，ここでは，捩じり運動の振幅を 0 としたので実際は意味なし．

なお，仮に，2m/s で遊泳しているトビウオ周りのレイノルズ数は，

水中 $\mathrm{Re}=\dfrac{UL}{\nu}=\dfrac{2*0.2}{1.0*10^{-6}}=4.0*10^{5}$

ストローハル数は，

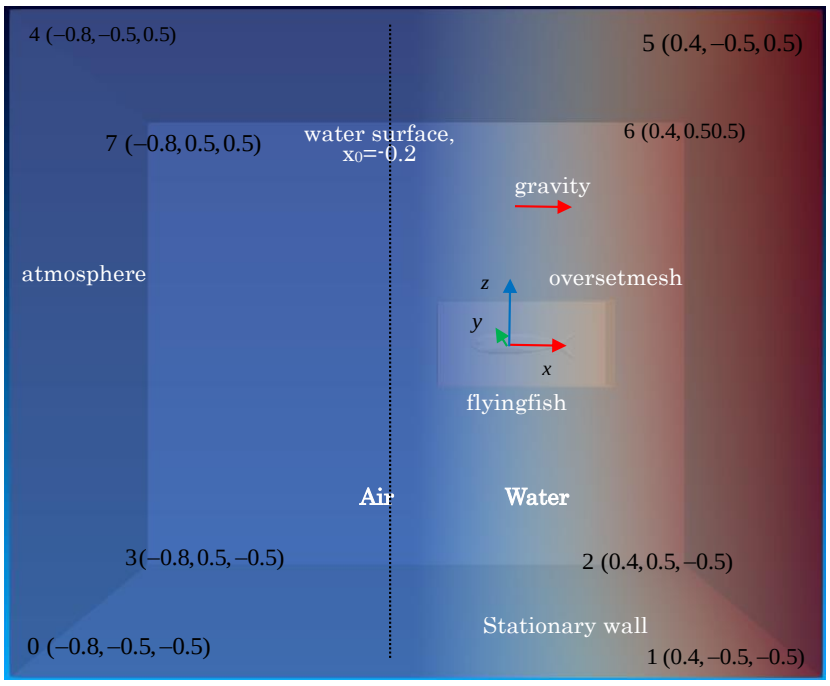
$\mathrm{Re}=\dfrac{fL}{U}=\dfrac{40*0.2}{2}=4$

ウェーバー数は，

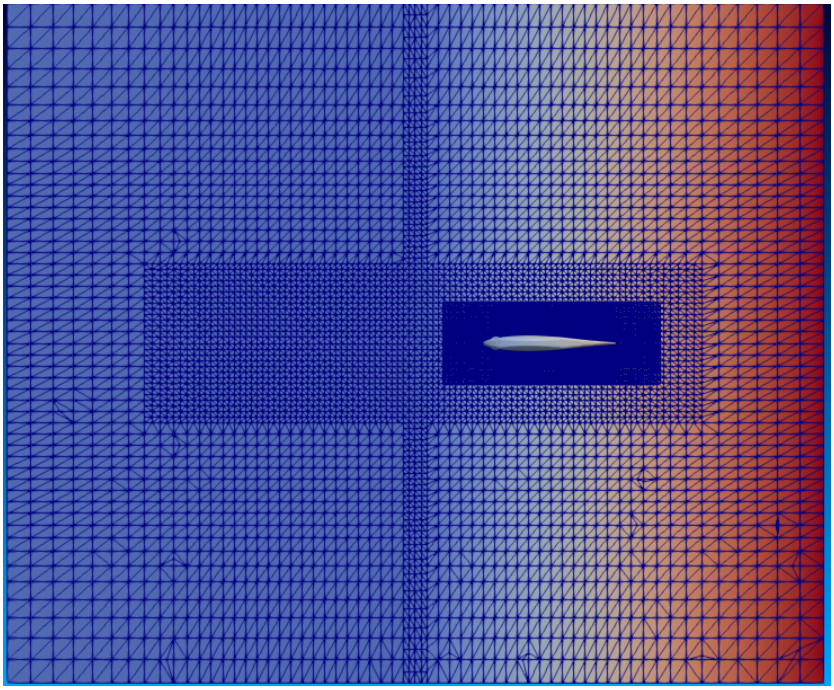
$\mathrm{Re}=\dfrac{\rho LU^{2}}{\sigma}=\dfrac{1000*0.2*2^{2}}{0.072}=1.1*10^{4}$

である．乱流であるが，空間的にはほとんど静止しているので一旦層流で計算してみる．

解析空間：図の通り，重力は x の方向（9.8, 0, 0）に働いており，水面は x=-0.2 である．これまでの水平遊泳のトビウオの CAD データを使いまわしただけで重力方向に特に意味はない．普通は，z の負の方向を重力方向にする．格子はとても粗くしてあるので，クラウンや水しぶきの再現はほとんどできていない．これらの解析には，格子をさらに少なくとも 2 段階（=1/4）は分割する必要がある．



解析空間



z=0 面の格子．とても粗い．水面，運動軌跡周りを 1 段階細分化している

境界条件は以下の通り．

境界条件

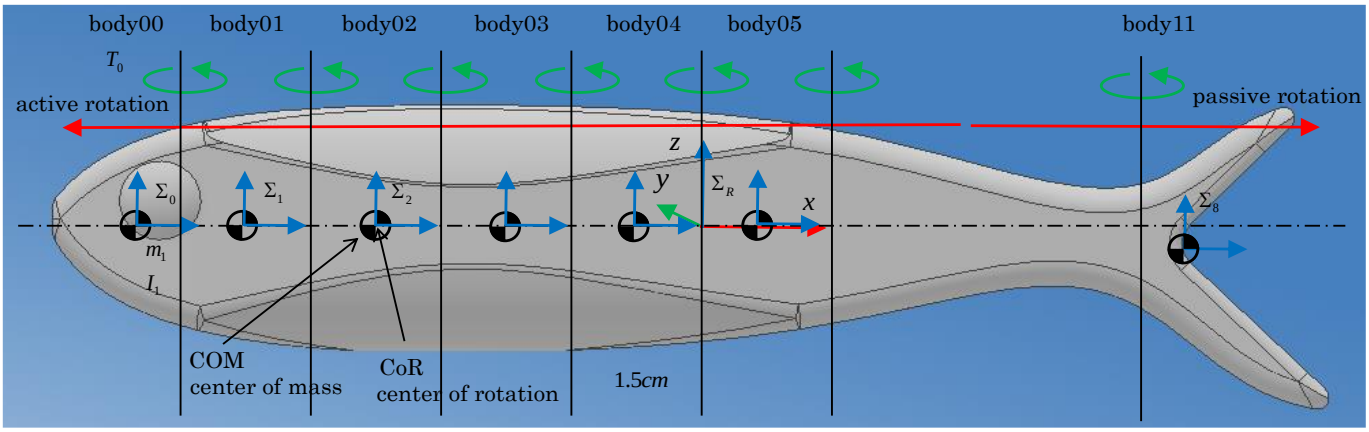
	Type	alpha.water	U	p_rgh	zoneID
atmosphere	patch	type inletOutlet; inletValue uniform 0; value uniform 0;	type pressureInletOutletVelocity; value uniform (0 0 0);	type totalPressure; p0 uniform 0; U U; phi phi; rho rho; psi none; gamma 1; value uniform 0;	type zeroGradient
StationaryWalls	wall	type zeroGradient;	type fixedValue; value uniform (0 0 0);	type fixedFluxPressure	type zeroGradient
body00-body11	wall	zeroGradient	type movingWallVelocity; value uniform (0 0 0);	type fixedFluxPressure;	type zeroGradient
fyingfishSides	overset				

	k	nut	omega	pointDisplacement
atmosphere				type fixedValue; value uniform (0 0 0)
StationaryWalls				type fixedValue; value uniform (0 0 0)
body00-body11				type calculated; value uniform (0 0 0);
flyingfishSides				patchType overset: type zeroGradient;

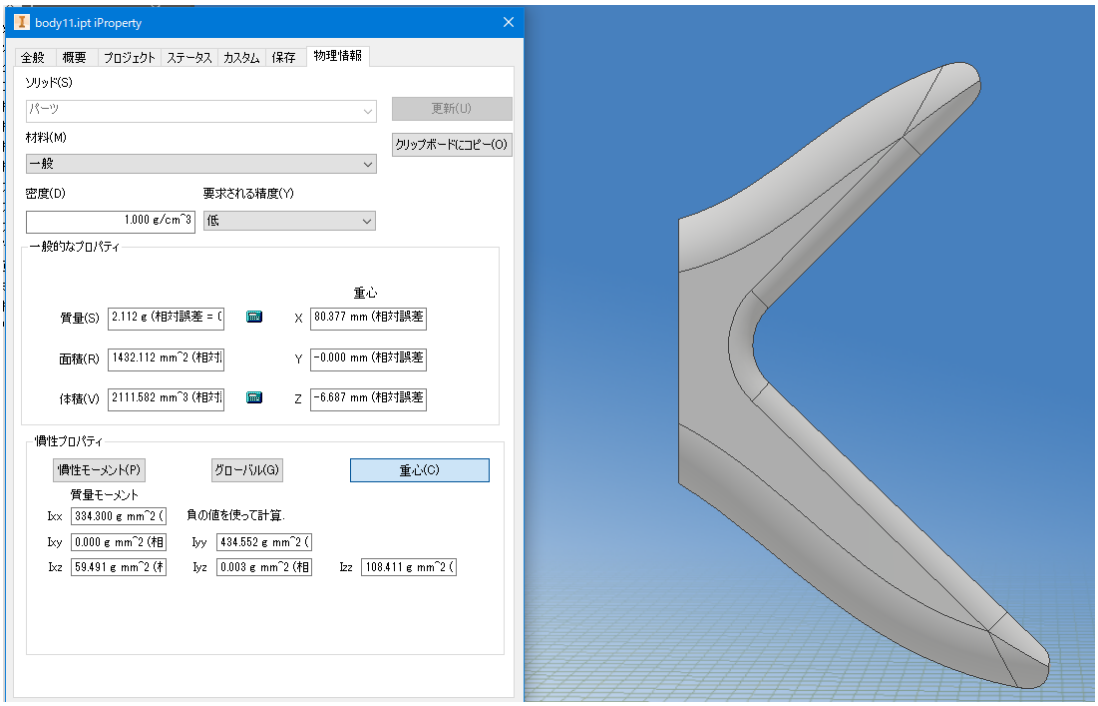
1. CAD モデルの生成

CAD でトビウオモデルを作図し，n 剛体に分解して stl ファイルで保存する（メートル，ASCII 設定）．CAD の原点が openFoam の原点となる．以下のディレクトリにファイルを保存しておく．

constant/triSurface/*.stl



ここでは，n=12 剛体でモデル化している．以下は，body11 の例．iProperty で，重心，質量，慣性テンソルを保存しておく．なお，主にヨー運動をするので，z 軸回りの慣性モーメント Izz が小さすぎると計算が安定しない．特に，隣り合う剛体どうしの慣性モーメントの 10 倍以上あると，硬い系と同じような状況になり，dt を小さくしても計算が安定し難くなる．



2. マルチボディによる遊泳運動

2.1 マルチボディによるトビウオの運動

それぞれの剛体は，並進と回転で 6 自由度で表現される．重心間が並進バネとダンパで結合されているが，回転に関しては受動的には結合されていない．ここでは，回転運動の式により全て目標値に対してトルクの PD 制御される．

2.2 尾びれ振り運動

トビウオ座標系の体軸を x 軸として，進行波をつくり，水を後ろに押し出して推進する．尾鰭側程二次関数的に振幅が大きくなる Carangiform である．これを表現する関数として，以下を用いる．

$$y(x,t) = (A_0 \frac{x}{L} + A_1 (\frac{x}{L})^2) \sin(2\pi(\frac{x}{\lambda} - ft)) \qquad (1)$$

体軸位置 x，時間 t に対して，側面変位 y が決まる．L は体長，λ は波長，f は周波数である．x=0 が頭部原点位置であり，A_0, A_1 が振幅係数で，A_1= 0 とすると，ウナギのような Anguilliform になる．ここでは，A_0=0.01 [m]，A_1=0.03 [m]としてあるが，実際のトビウオとは異なっている．また，L=λ=0.2 としている．体全体で一波長を作る．f は 20Hz にした．また，この式は，屈折と共に体長が長くなってしまう表現であるが，微小としてここでは無視している（後述するマルチボディによる回転運動では結局伸びない）．

運動制御は，マルチボディ間の目標角度に対してトルク PD 制御する．下図に示す通り，トビウオの頭部参照座標系に対して，剛体 i のヨー角 ψ i は，以下で表現できる．

$$\tan\psi_i = \frac{dy}{dx}$$

ここで,

$$\frac{dy}{dx} = (\frac{A_0}{L} + \frac{2A_1}{L^2}x)\sin(2\pi(\frac{x}{\lambda} - ft)) + (A_0\frac{x}{L} + A_1(\frac{x}{L})^2)\frac{2\pi}{\lambda}\cos(2\pi(\frac{x}{\lambda} - ft))$$

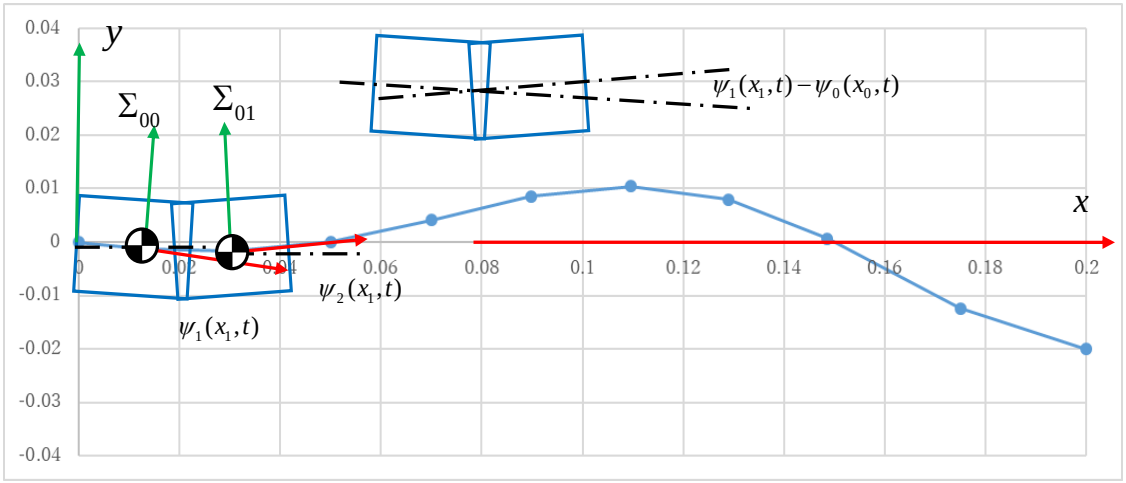
より,

$$\psi_i(x,t) = atan(\frac{dy}{dx})$$

であるので，剛体間の相対角度は,

$$\Delta\psi_i = \psi_i(x_i,t) - \psi_{i-1}(x_{i-1},t)$$

となる．これが，目標ヨー角度である．D 成分は，角速度ベクトルを使うため，後述するロドリゲス角から求める．



ロール運動は，尾びれ方向に進むほど一次的に振幅が大きくなる以下の式を用いる．ここでは，$B_0=0$ としたので実際は意味なし．ϕ は，(1)式に対する位相．概ね 90deg.

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

剛体間角度は,

$$\Delta\varphi_i = \varphi_i(x_i,t) - \varphi_i(x_{i-1},t)$$

となる.

以上より，P 偏差，D 偏差は，以下となる.

i-1 番目のボディから見た i 番目のボディの目標姿勢 ${}^{i-1}R_{i,t}$ は,

$${}^{i-1}R_{i,t} = R_z R_x = \begin{bmatrix} \cos\Delta\psi_i & -\sin\Delta\psi_i & 0 \\ \sin\Delta\psi_i & \cos\Delta\psi_i & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Delta\varphi_i & -\sin\Delta\varphi_i \\ 0 & \sin\Delta\varphi_i & \cos\Delta\varphi_i \end{bmatrix}$$

となる．このときの i-1 から見た i に姿勢を変更するロドリゲスの角度ベクトル $\boldsymbol{r}_i$ を

$$\boldsymbol{r}_i = f({}^{i-1}R_i)$$

とすると，現在の姿勢 ${}^{i-1}R_i$ との偏差角度ベクトルは,

$${}^{i-1}R_i \;\; f({}^{i-1}R_i^T \;\; {}^{i-1}R_{i,t})$$

となる．i-1 から見た i の目標角速度ベクトルは

$$\dot{\boldsymbol{r}}_i = {}^{i-1}R_{i,t-\Delta t}\frac{f({}^{i-1}R_{i,t-\Delta t}{}^T \;\; {}^{i-1}R_{i,t})}{\Delta t}$$

となるので，現在の速度ベクトルとの偏差角速度ベクトルは,

$${}^{i-1}R_{i,t-\Delta t}\frac{f({}^{i-1}R_{i,t-\Delta t}{}^T \;\; {}^{i-1}R_{i,t})}{\Delta t} - {}^{i-1}R\;\omega_{i,t}$$

となる．これより，PD 制御トルク T は,

$$T_{i,t} = P \quad {}^{i-1}R_i \quad f({}^{i-1}R_i^T \quad {}^{i-1}R_{i,t}) + D[\ {}^{i-1}R_{i,t-\Delta t} \frac{f({}^{i-1}R_{i,t-\Delta t}^T \quad {}^{i-1}R_{i,t})}{\Delta t} - {}^{i-1}R\ \omega_{i,t}]$$

となる．ここで，P, D はゲイン．

なお，この式は，トビウオの任意の姿勢に対して汎用化されていない．ピッチ姿勢＝ 0 deg の式．以下の COM03.7 ライブラリは，ピッチ角が**任意の初期姿勢**に対応していない．運動により変化するピッチ角には対応している．

2.2 プログラミング

COM03.7 以降が対応している．

．fishlocomotionCOM 関数が，上記運動に従い，マルチボディ間のトルクを制御する．

F:¥openFOAM¥lib¥COM03.7¥rigidBodyDynamics¥restraints¥fishlocomotionCOM

．fishlocomotionCOM.H

```
#ifndef RBD_restraints_fishlocomotionCOM_H
#define RBD_restraints_fishlocomotionCOM_H

#include "rigidBodyRestraint.H"

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

namespace Foam
{
namespace RBD
{
namespace restraints
{

/*---------------------------------------------------------------------------*¥
                        Class fishlocomotionCOM Declaration
¥*---------------------------------------------------------------------------*/

class fishlocomotionCOM
:
    public restraint
{
    // Private data

        // fish length [m]
        scalar      bodyLength_;

        // frequency [Hz]
        scalar      freq_;

        // wave length [m]
        scalar      lambda_;

        // caringiform parameter, A0 amplitude [m], A1 amplitude [m],    dummy
        vector      carangiform_;

        // rolling parameter, p0 amplitude [rad], p1 amplitude [rad], phase=0
        vector      rolling_;

       //- (P gain , D gain, max):
       vector gain_;

        // origin of fish coordinate system
        scalar      x_origin_;

        // relaxation time
       scalar relaxationTime_;

        //- body ID for the reaction force
        word        reaction_;
```

```
        //- report flg (default: false)
        const bool     fishlocomotionCOMReport_;

public:

    //- Runtime type information
    TypeName("fishlocomotionCOM");


    // Constructors

        //- Construct from components
        fishlocomotionCOM
        (
            const word& name,
            const dictionary& dict,
            const rigidBodyModel& model
        );

        //- Construct and return a clone
        virtual autoPtr<restraint> clone() const
        {
            return autoPtr<restraint>
            (
                new fishlocomotionCOM(*this)
            );
        }


    //- Destructor
    virtual ~fishlocomotionCOM();


    // Member Functions
        virtual void restrain
        (
            scalarField& tau,
            Field<spatialVector>& fx,
            const rigidBodyModelState& state
        ) const;

        //- Update properties from given dictionary
        virtual bool read(const dictionary& dict);

        //- Write
        virtual void write(Ostream&) const;

          // Transform Rodrigues matrix to Rodrigues vector
          vector        RodriguesVectorFromMatrix
          (
                const Tensor<scalar> R
          ) const;
};


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

} // End namespace restraints
} // End namespace RBD
} // End namespace Foam

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

#endif
```

. fishlocomotionCOM.C

```
#include "fishlocomotionCOM.H"
```

```
#include "rigidBodyModel.H"
#include "rigidBodyModelState.H"
#include "OneConstant.H"
#include "addToRunTimeSelectionTable.H"

//#include "rigidBodyMotion.H"

#define PI 3.141592654
#define      deg(a)        ((a)*180.0/PI)              // from rad to deg

// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
namespace RBD
{
namespace restraints
{
      defineTypeNameAndDebug(fishlocomotionCOM, 0);

      addToRunTimeSelectionTable
      (
            restraint,
            fishlocomotionCOM,
            dictionary
      );
}
}
}

// * * * * * * * * * * * * * * * * * Constructors    * * * * * * * * * * * * * * * //
Foam::RBD::restraints::fishlocomotionCOM::fishlocomotionCOM
(
      const word& name,
      const dictionary& dict,
      const rigidBodyModel& model
)
:
      restraint(name, dict, model),
      fishlocomotionCOMReport_( dict.getOrDefault<bool>("report", false))
{
      read(dict);
}

// * * * * * * * * * * * * * * * * * Destructor    * * * * * * * * * * * * * * * //
Foam::RBD::restraints::fishlocomotionCOM::~fishlocomotionCOM()
{}

// * * * * * * * * * * * * * * * Member Functions    * * * * * * * * * * * * * * * //
Foam::vector Foam::RBD::restraints::fishlocomotionCOM::RodriguesVectorFromMatrix
(
      const Tensor<scalar> R
) const
{
      vector n(0,0,0);
      scalar xx, yy, theta;
      yy = sqrt( sqr(R.xy()-R.yx()) + sqr(R.xz()-R.zx()) + sqr(R.yz()-R.zy()) ) / 2.0;
      xx = ( R.xx() + R.yy() + R.zz() - 1 ) / 2.0;

      theta = Foam::atan2( yy, xx );

      if( mag(theta) < (100*VSMALL) ){        // theta = 0 deg.
            n = Zero;
      }else if( mag( mag(theta) - mag(Foam::atan2(0.0,-1.0)) ) < (100*VSMALL) ){            // theta = +-180 deg.
            if( (R.xx()+1) < 0.0 )    n[0] = 0.0;
            else                                    n[0] = sqrt( (R.xx() + 1)/2.0 );
            if( (R.yy()+1) < 0.0 )    n[1] = 0.0;
            else{
```

```
            if( R.yx() > 0 )        n[1] = sqrt( (R.yy() + 1)/2.0 );
            else                    n[1] =-sqrt( (R.yy() + 1)/2.0 );
        }
        if( (R.zz()+1) < 0.0 )    n[2] = 0.0;
        else{
            if( R.zx() > 0 )        n[2] = sqrt( (R.zz() + 1)/2.0 );
            else                    n[2] =-sqrt( (R.zz() + 1)/2.0 );
        }
        n *= theta;

// Info << " theta*n " << n << endl;


    }else{
        n[0] = -(R.yz()-R.zy()) / (2.0*sin(theta));
        n[1] =    (R.xz()-R.zx()) / (2.0*sin(theta));
        n[2] = -(R.xy()-R.yx()) / (2.0*sin(theta));
        n *= theta;
    }

    return n;
}

void Foam::RBD::restraints::fishlocomotionCOM::restrain
(
    scalarField& tau,
    Field<spatialVector>& fx,
    const rigidBodyModelState& state
) const
{
//          Info << "fishlocomotionCOM: " << endl;

        label referenceID = model_.bodyID(reaction_);                   // reaction_ means a rigid body name
        const double A0 = carangiform_[0], A1 = carangiform_[1];
        const double p0 = rolling_[0], p1 = rolling_[1], phase = rolling_[2];
        const double t = state.t(), dt = state.deltaT();
        const double x = model_.I(bodyID_).c().x() - x_origin_, rx = model_.I(referenceID).c().x() - x_origin_;
        const double L = bodyLength_;

        // z: carangiform locomotion
        double      dydx = ( A0 / L + 2.0 * A1 / L / L * x ) *    sin( 2.0*PI*( x/lambda_ - freq_*t ) )
                        + ( A0 / L * x + A1 / L / L * x * x ) * 2.0*PI/lambda_ * cos( 2.0*PI*( x/lambda_ - freq_*t ) );
        double rdydx = ( A0 / L + 2.0 * A1 / L / L * rx ) * sin( 2.0*PI*( rx/lambda_ - freq_*t ) )
                        + ( A0 / L *rx + A1 / L / L *rx *rx ) * 2.0*PI/lambda_ * cos( 2.0*PI*( rx/lambda_ - freq_*t ) );
        double zangle = atan( dydx ) -    atan( rdydx );

                dydx = ( A0 / L + 2.0 * A1 / L / L * x ) *    sin( 2.0*PI*( x/lambda_ - freq_*(t-dt) ) )
                        + ( A0 / L * x + A1 / L / L * x * x ) * 2.0*PI/lambda_ * cos( 2.0*PI*( x/lambda_ - freq_*(t-dt) ) );
            rdydx = ( A0 / L + 2.0 * A1 / L / L * rx ) * sin( 2.0*PI*( rx/lambda_ - freq_*(t-dt) ) )
                        + ( A0 / L *rx + A1 / L / L *rx *rx ) * 2.0*PI/lambda_ * cos( 2.0*PI*( rx/lambda_ - freq_*(t-dt) ) );
        double preZangle = atan( dydx ) -    atan( rdydx );

        // x: body rolling
        double xangle = ( p0 / L * x + p1 / L / L * x * x ) * sin( 2.0*PI*( x/lambda_ - freq_*t ) + phase )
                          - ( p0 / L *rx + p1 / L / L *rx *rx ) * sin( 2.0*PI*(rx/lambda_ - freq_*t ) + phase );
        double preXangle = ( p0 / L * x + p1 / L / L * x * x ) * sin( 2.0*PI*( x/lambda_ - freq_*(t-dt) ) + phase )
                                - ( p0 / L *rx + p1 / L / L *rx *rx ) * sin( 2.0*PI*(rx/lambda_ - freq_*(t-dt) ) + phase );

        if( t < relaxationTime_    ){
                xangle *= (t/relaxationTime_);
                zangle *= (t/relaxationTime_);
                preXangle *= (t/relaxationTime_);
                preZangle *= (t/relaxationTime_);
        }


        Tensor<scalar> Rz, Rx, Rtarget, Rpre;
        Rz.xx() = cos(zangle);  Rz.yx() =-sin(zangle);  Rz.zx() = 0.0;
```

```
		Rz.xy() = sin(zangle);  Rz.yy() = cos(zangle); Rz.zy() = 0.0;
		Rz.xz() = 0.0;                  Rz.yz() = 0.0;                  Rz.zz() = 1.0;
		Rx.xx() = 1.0;                  Rx.yx() = 0.0;                  Rx.zx() = 0.0;
		Rx.xy() = 0.0;                  Rx.yy() = cos(xangle); Rx.zy() =-sin(xangle);
		Rx.xz() = 0.0;                  Rx.yz() = sin(xangle);  Rx.zz() = cos(xangle);
		Rtarget = Rz.T() & Rx.T();

		Rz.xx() = cos(preZangle);       Rz.yx() =-sin(preZangle);       Rz.zx() = 0.0;
		Rz.xy() = sin(preZangle);       Rz.yy() = cos(preZangle);       Rz.zy() = 0.0;
		Rz.xz() = 0.0;                          Rz.yz() = 0.0;                                  Rz.zz() = 1.0;
		Rx.xx() = 1.0;                  Rx.yx() = 0.0;                          Rx.zx() = 0.0;
		Rx.xy() = 0.0;                  Rx.yy() = cos(preXangle);       Rx.zy() =-sin(preXangle);
		Rx.xz() = 0.0;                  Rx.yz() = sin(preXangle);       Rx.zz() = cos(preXangle);
		Rpre = Rz.T() & Rx.T();

		/*** relative body angle ***/
		const Tensor<scalar> Rr = model_.X0(referenceID).E().T(), Rb = model_.X0(bodyID_).E().T();
		const Tensor<scalar> rRb = Rr.T() & Rb;

		const Tensor<scalar> bRtarget = rRb.T() & Rtarget; // angle error
		vector RodVector = RodriguesVectorFromMatrix( bRtarget );
		RodVector = rRb & RodVector;

		/*** relative body anglular velocity ***/
		const vector angular_vel = Rr.T() & ( model_.v(bodyID_).w() - model_.v(referenceID).w() );

		const Tensor<scalar> preRtarget = Rpre.T() & Rtarget;
		vector RodVVector = RodriguesVectorFromMatrix( preRtarget ) / dt;
		RodVVector = ( ( Rr.T() & Rpre ) & RodVVector ) - angular_vel;


		scalar	maxInertia = 0.0;
		for(int i=0; i<((model_.nBodies()-1)/2); i++){
			maxInertia = max( maxInertia, model_.I(i).Ic().zz() );
		}
		scalar weight = model_.I(bodyID_).Ic().zz() / maxInertia;

		vector moment = weight * ( gain_[0] * RodVector + gain_[1] * RodVVector );

//		if( t < relaxationTime_    )		moment *= (t/relaxationTime_);

		if( gain_[2] < mag( moment ) )	moment *= ( gain_[2] / mag( moment ) );


	if (model_.debug || fishlocomotionCOMReport_)
//	  if ( RBD::rigidBodyMotion::Iteration_number_for_MB_ == 0 )
	{
		Info<< " fishlocomotionCOM: " << endl
		<< model_.name(bodyID_) << "["<< bodyID_ << ", " << bodyIndex_ << "], " << reaction_ << "[" <<referenceID << "]" << endl
		<< "target z angle [deg] = " << deg(zangle) << " x angle [deg] = " << deg(xangle) << " ang.vel. [rad/s] = " <<
(RodriguesVectorFromMatrix( preRtarget ) / dt) << " , t= " << t << endl;
/*
Info << "R" << endl;
Info << Rtarget.xx() << " " << Rtarget.yx() << " " << Rtarget.zx() << endl;
Info << Rtarget.xy() << " " << Rtarget.yy() << " " << Rtarget.zy() << endl;
Info << Rtarget.xz() << " " << Rtarget.yz() << " " << Rtarget.zz() << endl;
Info << "Inertia body " << endl << model_.I(bodyID_).Ic().zz() << endl;
Info << "Inertia ref. " << endl << model_.I(referenceID).Ic().zz() << endl; */
Info << "Inertia weight " << weight << " " << maxInertia << " " << ((model_.nBodies()-1)/2) << endl;

		Info << "target angle [rad]: " << RodriguesVectorFromMatrix( Rtarget ) << " target angular velocity [rad/s]: " << ( ( Rr.T() & Rpre)
& (RodriguesVectorFromMatrix( preRtarget )/dt) ) << endl;
		Info << "current angle [rad]: " << RodriguesVectorFromMatrix( rRb ) << " current angular velocity [rad/s]: " << angular_vel <<
endl;

		Info << "target difference [rad]: " << RodVector << " target difference velocity [rad/s]: " << RodVVector << endl;
		Info << "moment: " << moment << " P " << weight * gain_[0] * RodVector << " D " << weight * gain_[1] * RodVVector    <<
endl;
```

```
Info << "gain " << gain_ << endl;
Info << "moment: " << weight * ( gain_[0] * RodVector + gain_[1] * RodVVector ) << endl;



            if( ( referenceID == 0 ) || ( bodyID_ == 0 ) )    Info << "The moment is not reacted to the root" << endl;
    }

    // Accumulate the ONLY MOMENT for the restrained body
    if( bodyIndex_ != 0 )           fx[bodyIndex_]    += spatialVector( Rr & moment, Zero /*force*/ );
    if( referenceID != 0 )          fx[referenceID] -= spatialVector( Rr & moment, Zero /*force*/ );
}


bool Foam::RBD::restraints::fishlocomotionCOM::read
(
    const dictionary& dict
)
{
    restraint::read(dict);

    coeffs_.readEntry("bodyLength", bodyLength_);
    coeffs_.readEntry("frequency", freq_);
    coeffs_.readEntry("waveLength", lambda_);
    coeffs_.readEntry("carangiform", carangiform_);
    coeffs_.readEntry("rolling", rolling_);
    coeffs_.readEntry("gain", gain_);
    coeffs_.readEntry("x_origin", x_origin_);
    coeffs_.readEntry("relaxationTime", relaxationTime_);
    coeffs_.readEntry("reaction", reaction_);

    return true;
}


void Foam::RBD::restraints::fishlocomotionCOM::write
(
    Ostream& os
) const
{
    restraint::write(os);

    os.writeEntry("bodyLength", bodyLength_);
    os.writeEntry("frequency", freq_);
    os.writeEntry("lambda", lambda_);
    os.writeEntry("carangiform", carangiform_);
    os.writeEntry("rolling", rolling_);
    os.writeEntry("gain", gain_);
    os.writeEntry("x_origin", x_origin_);
    os.writeEntry("relaxationTime", relaxationTime_);
    os.writeEntry("reaction", reaction_);
}
```

3. マルチボディに基づく動力学の設定

constant/dynamicMeshDict ファイルを用意する．

```
motionSolverLibs      (rigidBodyMeshMotionCOM);     // COM ソルバを選択

dynamicFvMesh           dynamicOversetFvMesh;         //  重合格子を選択

motionSolver            rigidBodyMotion;

report                  false;
meshMotionSmoothing 1.0e-7;    // マルチボディに基づく格子点の平滑化．デフォルトは，1.0e-15．運動によるボディ間の変位が大きいときには大きくする．とはいえ，これぐらいが限界．
```

```
solver
{
    type NewmarkCOMimplicit;            // for equation of motion around COM
    report off;
}
Iteration_number_for_MB         1;

OutputFiles             (body04body);       // save the center of rotation.  重心計算用ボディ．保存用.

bodies
{
    body00body
    {
        type            rigidBody;
        parent          root;           // parent always select "root" for equation of motion around COM.
        mass            0.00305;                    // rho = 1000 kg/m^3
        inertia         (166.9e-9 0 0.3e-9    170.3e-9 0    75.1e-9);         // COM: (166.9e-9 0 0.3e-9    170.3e-9 0    75.1e-9)
        centreOfMass    (-0.0899 0 0.0003);                                   /* global coordinate system */
        transform       (1 0 0    0 1 0    0 0 1) $centreOfMass;            // initial body coordinate system viewing from the parent
        joint
          {
            type            composite;
            joints (
                { type Pxyz; }
                { type Rxyz;        }    );
        }
        patches         (body00);
        innerDistance   0;
        outerDistance   101;
    }

    body01body
    {
        type            rigidBody;
        parent          root;
        mass            0.00822;
        inertia         (898.7e-9 0 6.1e-9    852.5e-9 0    349.2e-9);
        centreOfMass    (-0.0770 0 0);    /* global coordinate system */
        transform       (1 0 0    0 1 0    0 0 1) $centreOfMass;
        joint
          {
            type            composite;
            joints (
                { type Pxyz; }
                { type Rxyz;        }    );
        }
        patches         (body01);
        innerDistance   0;
        outerDistance   100;
    }

・・・・・

    body11body
    {
        type            rigidBody;
        parent          root;
        mass            0.0021;
        inertia         (334.3e-9 0 59.5e-9    434.6e-9 0    108.4e-9);        // Izz が小さくなりすぎないように注意する
        centreOfMass    (0.0804 0 -0.0067);          /* global coordinate system */
        transform       (1 0 0    0 1 0    0 0 1) $centreOfMass;
        joint
          {
            type            composite;
            joints (
                { type Pxyz; }
                { type Rxyz;        }    );
        }
```

```
            patches             (body11);
            innerDistance    0;
            outerDistance    100;
        }
}

// parameter definition
translational_spring    ( 4.0e+5 40 0 );        // spring coefficient, Damper coefficient, dummy
//coil_spring           ( 0.01 2.0e-6 0 );      // spring coefficient, Damper coefficient, dummy
output_gain             ( 50.0 1.0 100.0 );     // (P gain, D gain, max input)
carangiformPara ( 0.01 0.03 0 );    // A0, A1 [m], dummy
rollingPara ( 0 0 0 );              // p0, p1, phase [rad]
fishLength 0.2;         // [m]
lambda                  0.2;
tailFreq    20;         // [Hz]
relaxation_time 0.05;   // [s]
fishx_origin            -0.1;       // [m]


restraints
{
//    ********** body spring and damper *****************

    body00connection
    {
        type                    rigidbodyConnectionCOM;
        body                body00body;
          reaction    body01body;
          connection_point (-0.085 0 0);
        gain            $translational_spring;
    }

. . . .

    body10connection
    {
        type                    rigidbodyConnectionCOM;
        body                body10body;
          reaction    body11body;
          connection_point (0.07 0 0);
        gain            $translational_spring;
    }

//    ********** body coil-spring and damper *****************


//    ********** body control *****************

    body01control
    {
        type                    fishlocomotionCOM;
          reaction    body00body;             // body name for reaction torque
        body                body01body;
          gain                  $output_gain;           // (P gain, D gain, max input)
          bodyLength            $fishLength;
          frequency   $tailFreq;
          waveLength            $lambda;
          carangiform           $carangiformPara;
          rolling               $rollingPara;
          x_origin    $fishx_origin;
          relaxationTime        $relaxation_time;
//            report              on;
    }

. . . .

    body11control
    {
```

```
        type                    fishlocomotionCOM;
        reaction    body10body;         // body name for reaction torque
        body                body11body;
        gain                    $output_gain;           // (P gain, D gain, max input)
        bodyLength              $fishLength;
        frequency   $tailFreq;
        waveLength              $lambda;
        carangiform             $carangiformPara;
        rolling                 $rollingPara;
        x_origin    $fishx_origin;
        relaxationTime          $relaxation_time;
//      report                  on;
    }
}
```

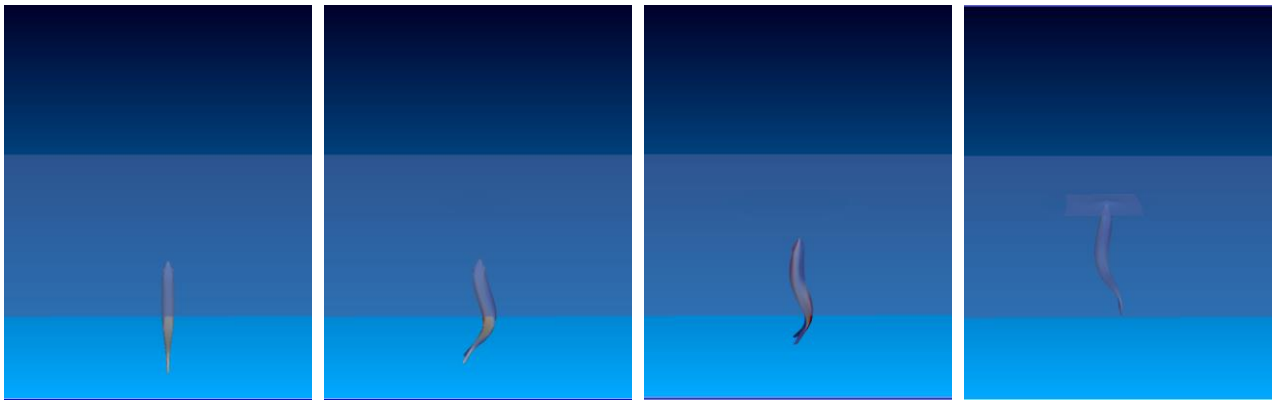
4. 解析用のモーメント計算用の点の記述

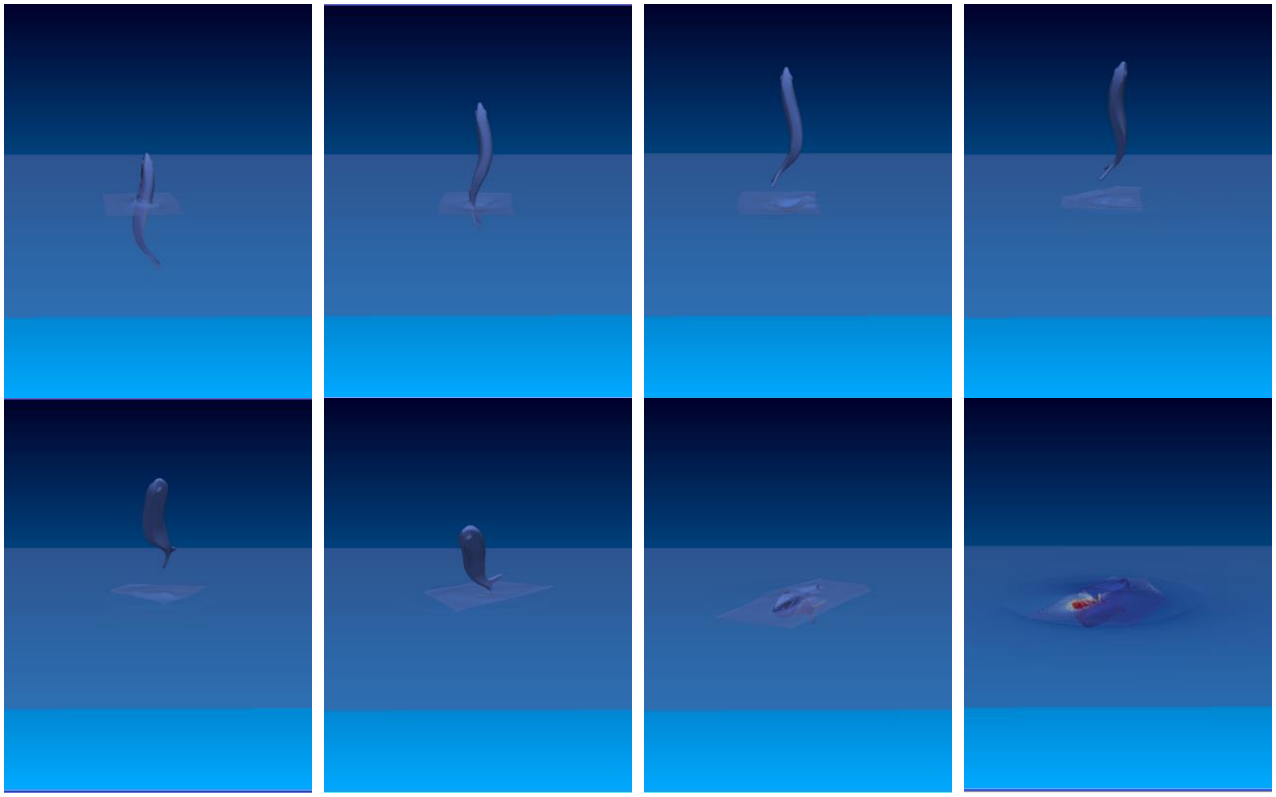
system/controlDict ファイルに流体力によるモーメントを計算する点をいくつか設定しておく．CAD の iProperty で重心を確認しておく．

```
functions
{
    fishforce.dat
    {
     type            forces;
     libs            ("libforcesCOM.so");
     patches         (body00 body01 body02 body03 body04 body05 body06 body07 body08 body09 body10 body11);      // 圧力を積分する境界．これが外力になる．

     writeControl        runTime;
     writeInterval       0.001;

//   rho             rhoInf;
//   rhoInf          998;            // water ?

     cofRname        body04body;     // Open this body file and read the center of rotation. this file will be updated every time step.
     CofR            (0.0109 0 -0.0002);     // (-0.0216 0 -0.0003)
                                     body04 の重心が(-0.0325 0 -0.0001)であるので，(-0.0325 0 -0.0001) + (0.0109 0 -0.0002) = (-0.0216 0 -0.0003)の重心周りを計算していることになる．
     log             false;
    }
}
```

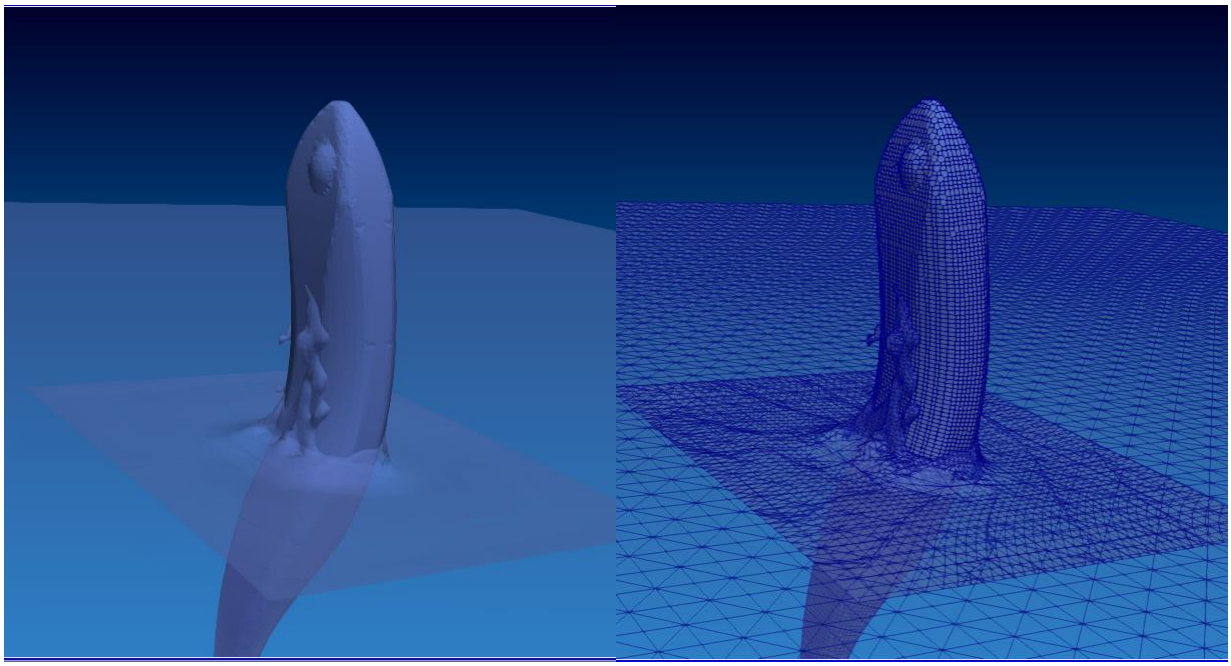
5. 実行

以下，0.06 秒毎時系列に表示する．0.61 で計算が破綻している．原因は不明．そもそも計算が粗い．





離水時の水面の状態．粗すぎて水しぶき，クラウン他，表現できていない．



姿勢を崩して着水しても水しぶき出ず．．．．．
恐らく，さらに格子を 1/4 ぐらいにしないと，正しく計算できない．