Basilisk On-water Run by openFoam v2006

2021, August 17 修正

重合移動格子（oversetmesh）を用いて混相流（multiphase flow）における運動する物体周りの流体計算（一方向連成計算）を行う．図に示すように，水槽内（上面が空気，その他は壁）を仮定し，バシリスクが水面で脚を動かしたときの周りの流体計算．構造側のバシリスクは，剛体として定義されているが，境界面を移動させているので連続体として運動する．ただし，流体の反力は反映さず，予め決められた運動（運動学）をする．流体側は，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．変形，移動するセルに密度比を与えて計算している．例えば，水のセルは 1（=1000 [kg/m^3]），空気は 0（=1.3 [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，0.5 になる．また，重合格子を使っている．ここで，重合格子の形状を脚構造に合わせ，大変形を可能にしている．

以下，計算条件

モデル：　OpenFoam の定義上では 1 剛体．ライブラリにおいて，脚の境界面を移動させ，連続体を表現する．脚は 3 自由度のリンク機構と同じ．

運動：位相の 180deg 異なる左右脚を運動学に基づき，各 3 自由度（y 回転：θ_{0L}，θ_{1L}，θ_{2L}，θ_{0R}，θ_{1R}，θ_{2R},）で動かす．流体の反力は返らない．f= 10 [Hz]．左右の位相差は 180deg．振幅は 30deg.

$$\theta_{0i} = \hat{\theta}_{0i} \sin(2\pi t)$$

$$\theta_{1i} = \hat{\theta}_{1i} (\sin(2\pi t) - 1)$$

$$\theta_{2i} = \hat{\theta}_{2i} (\sin(2\pi t) + 1)$$

レイノルズ数は，3*0.4=1.2m の脚が 10Hz で 30deg の振幅と想定し，水上移動速度が最終的に足振り速度の 50%に達すると仮定すると，

水中　：$\mathrm{Re} = \dfrac{UL}{\nu} = \dfrac{(0.12*4*30/180*\pi*10*0.5)*(0.12)}{1.0*10^{-6}} = 1.5*10^5$

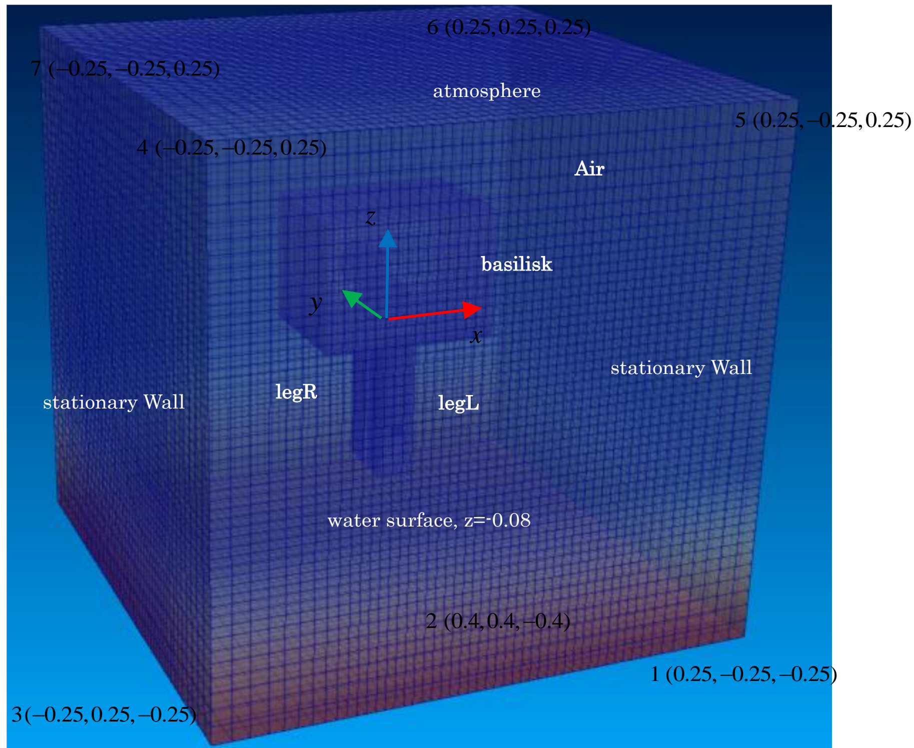
ストローハル数は，

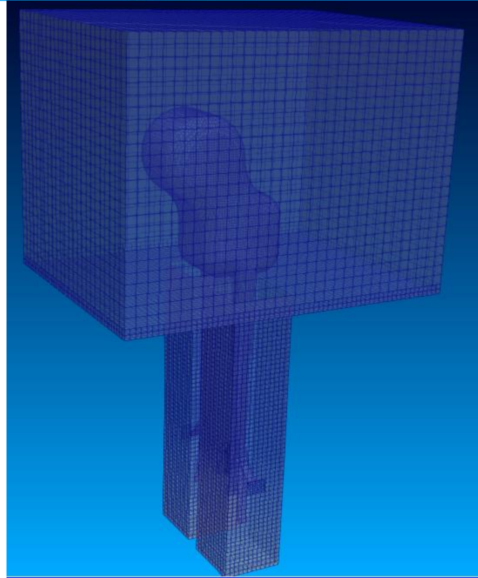
$$\mathrm{Re} = \frac{fL}{U} = \frac{10*0.12}{0.12*30*4/180*\pi*10*0.5} = 0.95$$

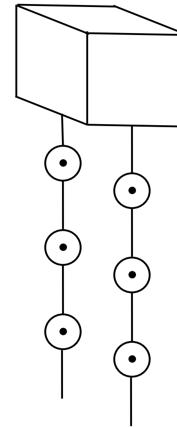
ウーバー数は，

$$We = \frac{\rho L U^2}{\sigma} = \frac{1000*0.12*(0.12*30*4/180*\pi*10*0.5)^2}{0.072} = 2.6*10^3$$

である．マッハ数も低いので，非定常，非圧縮，乱流ということになるが，ここでは，定常速度に達する前の水上移動速度が 0m/s に近い状態のシミュレーションを行うため，境界条件は層流と仮定して以下表のとおりとする．なお，天井が大気，それ以外は壁．ボディの中心を原点として，水面は z=-0.08m である．水面のスラップによるクラウンを表現したいところであるが，ここでは，格子は粗めに切ってある．







boundary name	Type	alpha.water	U 速度	p_rgh 圧力と水圧	pointDisplacement 運動で与える変位	Zone ID
atmosphere	patch	type inletOutlet; inletValue uniform 0; value uniform 0;	type pressureInletOutletVelocity; value uniform (0 0 0);	type totalPressure; p0 uniform 0;	type fixedValue; value uniform (0 0 0);	type zeroGradient
stationaryWalls	wall	zeroGradient	type fixedValue; value uniform (0 0 0);	type fixedFluxPressure;	type fixedValue; value uniform (0 0 0);	type zeroGradient
****Sides	overset			type zeroGradient;	patchType overset; type zeroGradient;	
robobody, legR, legL, footR, footL	wall	zeroGradient	type movingWallVelocity; value uniform (0 0 0);	type fixedFluxPressure;	type calculated; value uniform (0 0 0);	type zeroGradient
oversetPatch	overset			type overset;	patchType overset; type zeroGradient;	

overset				patchType overset; type fixedFluxPressure;		

注：slip は，スカラー値の場合には zeroGradient，ベクトル量の場合には，法線方向が zero,接線方向が zeroGradient とのこと．記述無しは，#includeEtc "caseDicts/setConstraintTypes"で読み込んでいるようだ．

1. Tutorial とライブラリのコピー

ライブラリ COM02 以降を利用する．実際に書き換えて使っているのは，rigidBodyMeshMotion から呼び出されている rigidBodyDynamics からさらに呼び出されている rigidBodyMotion クラスの transformPoints 関数のみ．定義された剛体が単数の時と複数のときで二つの関数が overload されているが，ここでは単数の方の関数を使う．

lib¥rigidBodyDynamics¥rigidBodyMotion¥rigidBodyMotion.H

ヘッダファイル．新しい変数を追加している．関節角度の振幅値など，汎用化する必要があれば追加する．

```
class rigidBodyMotion
:
    public rigidBodyModel
{
    friend class rigidBodySolver;

    // Private data

....

        //- Switch to turn FSI calculation on and off   以下，連続体格子移動用パラメータ
        Switch FSI_;            // FSI or oneway-FSI  フラグ   true は双方向連成

         vector     joint0_;           //  関節位置  (x y z) [m]，  以下，  y は常に 0
         vector     joint1_;           //  関節位置  position (x y z) [m]
         vector     joint2_;           //  関節位置  position (x y z) [m]
         scalar     motionFreq_;                 // motion frequency [Hz]
         scalar     relaxation_time_;  // relaxation time for stable calculation [s]  緩和時間
```

lib¥rigidBodyDynamics¥rigidBodyMotion¥rigidBodyMotion.C

プログラム．ここに脚の運動学を記述する．

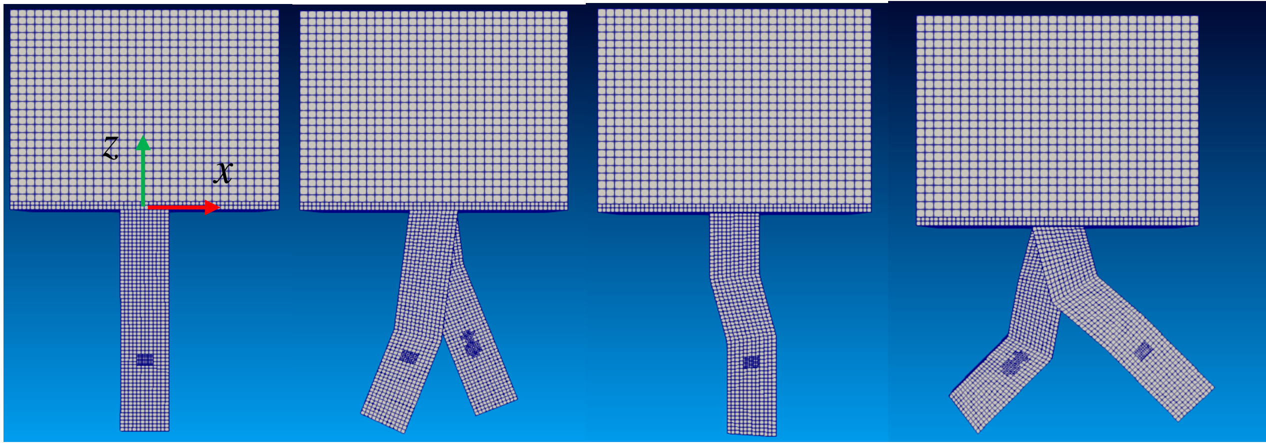
```
Foam::RBD::rigidBodyMotion::rigidBodyMotion
(
    const Time& time,
    const dictionary& dict
)
:
    rigidBodyModel(time, dict),
    motionState_(*this, dict),
    motionState0_(motionState_),
    X00_(X0_.size()),
    aRelax_(dict.getOrDefault<scalar>("accelerationRelaxation", 1)),
    aDamp_(dict.getOrDefault<scalar>("accelerationDamping", 1)),
    report_(dict.getOrDefault<Switch>("report", false)),
    FSI_(dict.getOrDefault<Switch>("FSI", true)),         //  一方向，双方向切り替えフラグ
    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{          // 追加変数の読み込み
        dict.readEntry("joint0", joint0_);
        dict.readEntry("joint1", joint1_);
        dict.readEntry("joint2", joint2_);
        dict.readEntry("motionFreq", motionFreq_);
        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{          // 追加ライブラリの読み込み
        dict.readEntry("joint0", joint0_);
        dict.readEntry("joint1", joint1_);
        dict.readEntry("joint2", joint2_);
        dict.readEntry("motionFreq", motionFreq_);
        dict.readEntry("relaxation_time", relaxation_time_);
    }
}


Foam::tmp<Foam::pointField> Foam::RBD::rigidBodyMotion::transformPoints
(
    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 rountine ***  双方向連成のルーチン
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 rountine ***    一方向連成のルーチン
Info << " rigidBodyMotion::transformPoints oneway-FSI " << endl;

            const scalar t = state().t();           //  現在時間
            scalar      theta0L = rad(30.0) * sin( 2.0 * PAI * motionFreq_ * t );  //  各関節角度
            scalar      theta0R = rad(30.0) * sin( 2.0 * PAI * motionFreq_ * t + rad(180.0) );
            scalar      theta1L = rad(30.0) * ( sin( 2.0 * PAI * motionFreq_ * t ) - 1.0 );
            scalar      theta1R = rad(30.0) * ( sin( 2.0 * PAI * motionFreq_ * t + rad(180.0) ) - 1.0 );
            scalar      theta2L = rad(30.0) * ( sin( 2.0 * PAI * motionFreq_ * t ) + 1.0 );
            scalar      theta2R = rad(30.0) * ( sin( 2.0 * PAI * motionFreq_ * t + rad(180.0) ) + 1.0 );
            const scalar link0 = mag( joint0_.z() - joint1_.z() );
            const scalar link1 = mag( joint1_.z() - joint2_.z() );

            if( t < relaxation_time_ ){             //  計算安定のための緩和時間
                    theta0L *= ( t / relaxation_time_ );
                    theta1L *= ( t / relaxation_time_ );
                    theta2L *= ( t / relaxation_time_ );
                    theta0R *= ( t / relaxation_time_ );
                    theta1R *= ( t / relaxation_time_ );
                    theta2R *= ( t / relaxation_time_ );
            }

            Tensor<scalar> R0L, R1L, R2L, R0R, R1R, R2R;          //  回転行列定義
//  以下，各関節の回転行列．行と列が入れ替わっているので注意する．ロボティクスは縦ベクトル，OpenFoam  は横ベクトル．
            R0L.xx() = cos(theta0L);        R0L.xy() = 0.0;     R0L.xz() = sin(theta0L);        // waring:: inversed
            R0L.yx() = 0.0;                 R0L.yy() = 1.0;     R0L.yz() = 0.0;
            R0L.zx() =-sin(theta0L);        R0L.zy() = 0.0;     R0L.zz() = cos(theta0L);

            R1L.xx() = cos(theta1L);        R1L.xy() = 0.0;     R1L.xz() = sin(theta1L);        // waring:: inversed
            R1L.yx() = 0.0;                 R1L.yy() = 1.0;     R1L.yz() = 0.0;
            R1L.zx() =-sin(theta1L);        R1L.zy() = 0.0;     R1L.zz() = cos(theta1L);
```

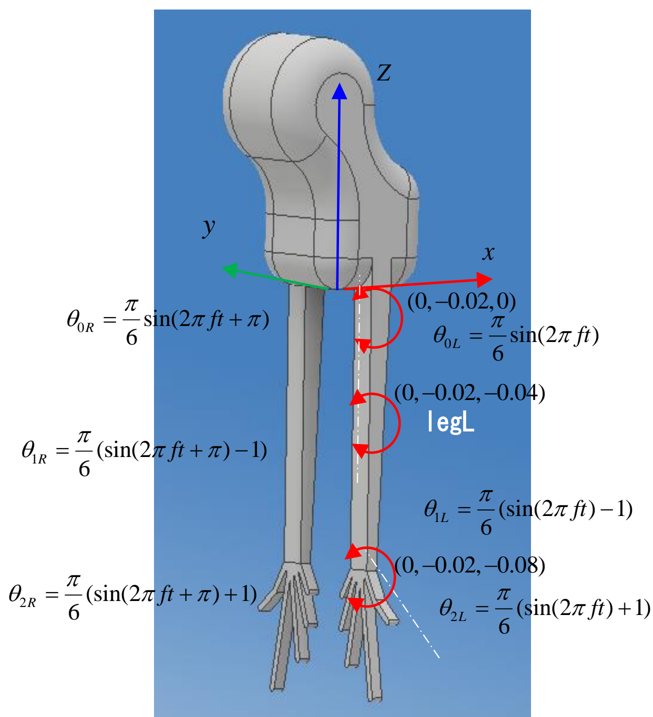
```
        R2L.xx() = cos(theta2L);        R2L.xy() = 0.0;     R2L.xz() = sin(theta2L);        // waring:: inversed
        R2L.yx() = 0.0;                 R2L.yy() = 1.0;     R2L.yz() = 0.0;
        R2L.zx() =-sin(theta2L);        R2L.zy() = 0.0;     R2L.zz() = cos(theta2L);

        R0R.xx() = cos(theta0R);        R0R.xy() = 0.0;     R0R.xz() = sin(theta0R);        // waring:: inversed
        R0R.yx() = 0.0;                 R0R.yy() = 1.0;     R0R.yz() = 0.0;
        R0R.zx() =-sin(theta0R);        R0R.zy() = 0.0;     R0R.zz() = cos(theta0R);

        R1R.xx() = cos(theta1R);        R1R.xy() = 0.0;     R1R.xz() = sin(theta1R);        // waring:: inversed
        R1R.yx() = 0.0;                 R1R.yy() = 1.0;     R1R.yz() = 0.0;
        R1R.zx() =-sin(theta1R);        R1R.zy() = 0.0;     R1R.zz() = cos(theta1R);

        R2R.xx() = cos(theta2R);        R2R.xy() = 0.0;     R2R.xz() = sin(theta2R);        // waring:: inversed
        R2R.yx() = 0.0;                 R2R.yy() = 1.0;     R2R.yz() = 0.0;
        R2R.zx() =-sin(theta2R);        R2R.zy() = 0.0;     R2R.zz() = cos(theta2R);

    forAll(points, i)   //  全格子ルーチン
    {
        const vector cpos = initialPoints[i] - joint0_;   //  初期格子位置を関節 1 座標系に並進変換.
        vector pos = initialPoints[i] - joint0;
        if( ( weight[i] > SMALL ) && ( cpos.z() < -0.001 ) ){      //  重合格子領域（weight[i]>0）かつ脚領域（z<0）だったら
          if( cpos.y() < 0.0 ){  // *** for legL  左足.
            if( joint1_.z() < cpos.z() ){                 // first link    リンク 0
                pos.z() += cpos.x()*tan(theta0L       ) * (cpos.z()-joint1_.z())/link0;
                pos.z() += cpos.x()*tan(theta1L/2.0) * cpos.z()/link0;
            }else if( joint2_.z() < cpos.z() ){           // second link       リンク 1
                pos.z() += cpos.x()*tan(theta1L/2.0) * (cpos.z()-joint2_.z())/link1;
                pos.z() += cpos.x()*tan(theta2L/2.0) * (cpos.z()-joint1_.z())/link1;
              pos = ( R1L & ( pos - joint1_ ) ) + joint1_;
            }else{                                        // third link       リンク 2
                pos.z() += cpos.x()*tan(theta2L/2.0) * (cpos.z()+0.12)/0.04;
              pos = ( R2L & ( pos - joint2_ ) ) + joint2_;
              pos = ( R1L & ( pos - joint1_ ) ) + joint1_;
            }
            pos = ( R0L & pos );
          }else{                              // *** for legR    右脚
            if( joint1_.z() < cpos.z() ){
                pos.z() += cpos.x()*tan(theta0R       ) * (cpos.z()-joint1_.z())/link0;
                pos.z() += cpos.x()*tan(theta1R/2.0) * cpos.z()/link0;
            }else if( joint2_.z() < cpos.z() ){
                pos.z() += cpos.x()*tan(theta1R/2.0) * (cpos.z()-joint2_.z())/link1;
                pos.z() += cpos.x()*tan(theta2R/2.0) * (cpos.z()-joint1_.z())/link1;
              pos = ( R1R & ( pos - joint1_ ) ) + joint1_;
            }else{
                pos.z() += cpos.x()*tan(theta2R/2.0) * (cpos.z()+0.12)/0.04;
              pos = ( R2R & ( pos - joint2_ ) ) + joint2_;
              pos = ( R1R & ( pos - joint1_ ) ) + joint1_;
            }
            pos = ( R0R & pos );
          }

         points[i] = pos + joint0_;
        }
    }
  }
    return tpoints;
}
```

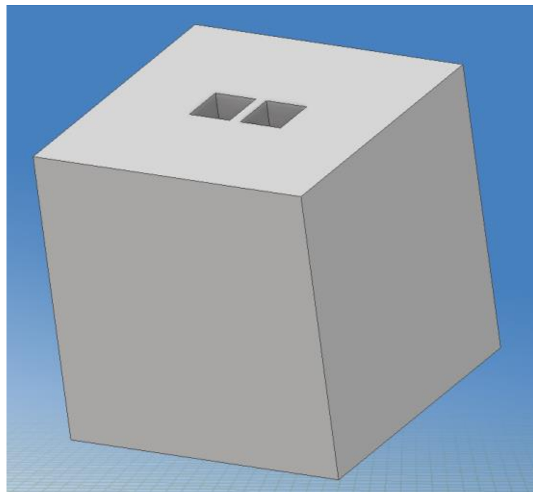
これより，下図のような領域返変換になる.



2. CAD モデルと運動重合格子の計算空間の生成

CAD で剛体モデルを作成し，stl ファイル（"basilisk.stl”）にする（mm 設定，ASCII 設定に注意）．なお，ここでは，重合格子の領域形状を変更するためだけの CAD モデル（"cutter.stl”）も作成している．これにより，左右脚領域を部分的に分割している．左右の脚の領域を切り離すことで，格子の歪を回避している．上部とはつながっている．





3. 各重合格子の計算空間の生成

上記剛体モデルの重合格子の計算空間を生成する．

oversetmesh/constant/triSurface/

というディレクトリを作成し，CAD で作ったボディのふたつのモデルをここにコピーする．

． oversetmesh/system/blockMeshDict

重合格子用の計算空間の設定ファイルを変更する．なお，blockMeshDict とは，bolockMesh 関数が呼び出す辞書（dictionary）という意味である．

```
scale    1;

vertices
(
```

```
        (-0.08 -0.06 -0.14)          // 下の面の座標４点．ナンバリングは０からこの順番
        ( 0.08 -0.06 -0.14)
        ( 0.08    0.06 -0.14)
        (-0.08    0.06 -0.14)

        (-0.08 -0.06 0.12)           // 上の面
        ( 0.08 -0.06 0.12)
        ( 0.08    0.06 0.12)
        (-0.08    0.06 0.12)
);

blocks
(
        hex (0 1 2 3 4 5 6 7) basiliskZone (32 24 56) simpleGrading (1 1 1)        // 形状指定，ゾーン名，xyz 分割数，分割率
);

edges
(
);

boundary
(
        basiliskSides      // 重合格子側面境界
        {
                type overset;
                faces
                (
                        (0 3 2 1)
                        (2 6 5 1)
                        (1 5 4 0)
                        (3 7 6 2)
                        (0 4 7 3)
                        (4 5 6 7)
                );
        }

        cutterSides                  // CAD モデルで切り取った面の境界名．この概念により複雑な形状を作れるようになった
        {
                type overset;        // 重合格子境界
                faces ();
        }

        basilisk                     // バシリスク境界名
        {
                type wall;
                faces ();
        }
);
```

. **snappyHexMeshDict**

```
castellatedMesh true;
snap                 true;
addLayers            false;

geometry
{
        basilisk            // バシリスク境界
        {
                type       triSurfaceMesh;
```

```
        file        "basilisk.stl";
    }
    cutterSides         // 切り取った重合格子の境界
    {
        type        triSurfaceMesh;
        file        "cutter.stl";
    }
    refinementBox    // 細分化領域指定
    {
         type                    searchableBox;
         min                      (-0.1 -0.2 -0.14);
         max                      ( 0.1    0.2 -0.08);
    }
};

castellatedMeshControls
{

    maxLocalCells 100000;
    maxGlobalCells 2000000;
    minRefinementCells 10;
    nCellsBetweenLevels 2;

    features ();

    refinementSurfaces
    {
        basilisk
        {
            // Surface-wise min and max refinement level
            level (2 2);            // 分割レベル
        }
        cutterSides
        {
            // Surface-wise min and max refinement level
            level (1 1);
        }
    }

    // Resolve sharp angles on fridges
    resolveFeatureAngle 30;

    refinementRegions
    {
         refinementBox
        {
            mode inside;
            levels ((1 1));        // 外の格子が細かければ，もっと切りたい．
        }
    }

    locationInMesh (0 0 0.1);      // 残す方の領域の点指定

    allowFreeStandingZoneFaces true;
}
```

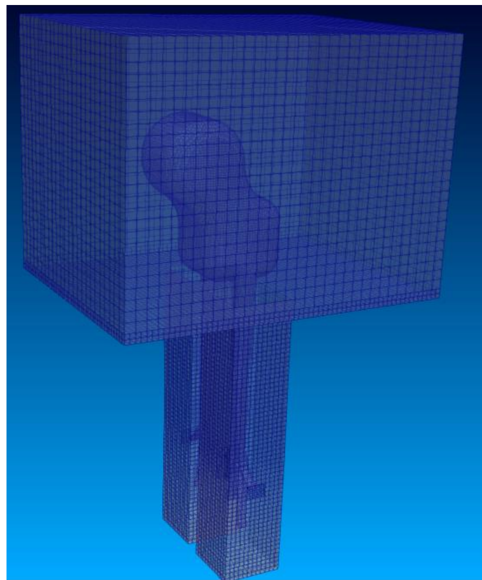
・・・

system 内のこれ以外のファイルはそのまま．なお，この時点で，linux において

```
> blockMesh
```

> snappyHexMesh –overwrite

を実行して paraview で確認すると，以下のとおり．下側の足の周りが，cutter.stl で切断した領域で，この面も重合格子境界になっている．



4. 外側計算空間の設定

一つ上のディレクトリに移動し，system 内のファイルを変更して外側計算空間を生成する．ここは別段記載事項無し．別資料参照．

5. 物性・運動設定（constant ディレクトリファイル）の修正

足先の運動学に関しても，別資料参照．菊池研生体工学など．

. constant/dynamicMeshDict

剛体の物性と運動指定ファイル．FSI 変数を"false"に設定した時点で，動力学は全て無視される．物体表面の移動のみ行われる．

```
dynamicFvMesh          dynamicOversetFvMesh;        // 重合格子を使う宣言

motionSolverLibs       (rigidBodyMeshMotionCOM);    // 動力学ライブラリ．COM ver 2.0 以降を使うこと
motionSolver           rigidBodyMotion;             // ソルバは，実は rigidBodyMeshMotion を使っている

report                 off;
rho rhoInf;
rhoInf 1.293;
g                      ( 0 0 -9.8065 );

solver
{
    type NewmarkCOM;            // 積分法はニューマークベータ
//      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          (body);      // save the center of rotation.

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

bodies
```

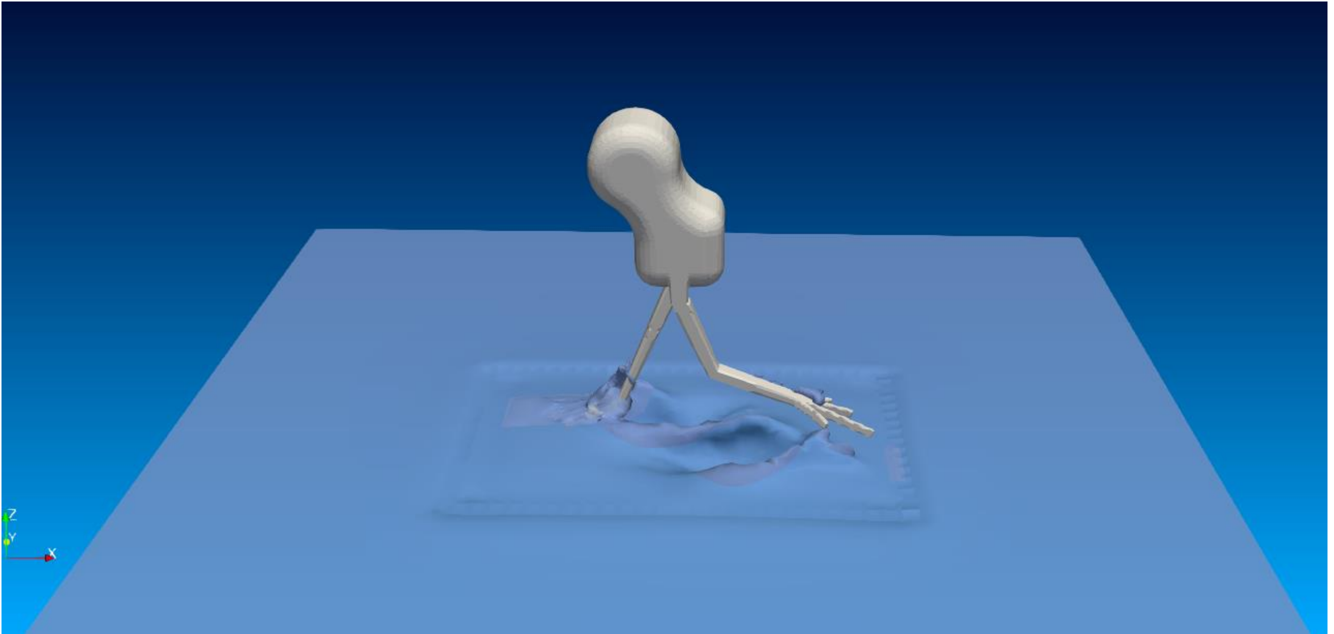
{ 　　　// ここは重要．剛体は必ずひとつのみ．openfoam では，剛体が単数か，複数かで関数がオーバーロードされている．ここでは，単数の関数を使っている．重心，慣性，質量などのパラメータは使っていない．

```
    basiliskbody
    {
        type                rigidBody;
        parent              root;
        mass                0.100;
        inertia             (100000e-9 0 0    100000e-9 0    10000e-9);
        centreOfMass        (0 0 0.01);                     /* global coordinate system */
        transform           $golocalCoordinateSystem $centreOfMass;
        joint {
            type                composite;
            joints (
                {   type Pxyz;          }
                {   type Rxyz;          }
            );
        }
        patches             (basilisk);
        innerDistance       0;
        outerDistance       101;
    }
}

restraints
{
}

FSI         false;      // これにより，oneway FSI を選択する．これ以降，動力学は無視される．

// ******* flyingfish parameters *******
joint0                  (0 0 0);        // joint position (x, y, z) [m], y is always zero. 関節位置指定． y は常に 0
joint1                  (0 0 -0.04);// joint position (x, y, z) [m], y is always zero.
joint2                  (0 0 -0.08);// joint position (x, y, z) [m], y is always zero.

motionFreq10;           // wave motion frequency [Hz]

relaxation_time 0.1;    // relaxation time for stable calculation
```

6. 実行

Librun でライブラリを作り，Allrun で実行する．ライブラリは，＊＊＊COM を消して上書きしているので，別プログラムと競合しているときには注意すること．

他の計算とたがわず，精度はわるい．クラウンなどは全く表現できていない．格子サイズは，クラウンを表現するためには，3level 上げる（1/2*1/2*1/2=1/8）必要がある．


Z
Y
X